NP-completeness

Lecture in INF4130

Department of Informatics

November 1st, 2018

Recap from Lecture 1 & 2

- Undecidability: no Turing Machine decides L
- Proving undecidability
  - First, we proved that the Halting problem is undecidable
  - Later, we proved more undecidability-results via *reductions*

Recap from Lecture 1 & 2

- Undecidability: no Turing Machine decides L
- Proving undecidability
  - First, we proved that the Halting problem is undecidable
  - Later, we proved more undecidability-results via *reductions*
- Defined running times for DTMs and NTMs
- Defined the complexity classes P and NP
- We also defined polynomial time reductions, but did not spend much time on them
- Briefly looked at the hierarchy of complexity classes

Today

- Define the notion of *NP*-completeness
- The *NP*-complete problems will be the "hardest" problems in *NP*
- We will see that *NP*-complete problems exist, by looking at a proof showing that a particular problem is *NP*-complete "from scratch" (like we did for undecidability with halting)
- Then we will show the *NP*-completeness of other problems via (polynomial time) reductions
- We will try to build up a hierarchy of *NP*-complete problems

Repetition: Polynomial time reductions

Definition (Polynomial reductions)

Language A is *polynomial time reducible* to language B, written $A \leq_P B$, if there exists a polynomial time computable function $f : \Sigma^* \to \Sigma^*$, where for every w:
$w \in A \leftrightarrow f(w) \in B$. The function f is called the polynomial (time) reduction from A to B.

NP-completeness

Definition (NP-completeness)

A language L is NP-complete if the two following hold for L:

1. $L \in NP$
2. for any language A in NP, $A \leq_P L$.

If L merely fulfills property (2), we call it NP-hard.

Theorem

If A is NP-complete and $A \in P$, then $P = NP$.

Theorem

If A is NP-complete and $A \in P$, then $P = NP$.

The theorem serves as an indication that NP-complete problems probably cannot be solved in polynomial time. It also partly explains why we are interested in proving that problems are NP-complete.

Theorem

If A is NP-complete and $A \in P$, then $P = NP$.

The theorem serves as an indication that NP-complete problems probably cannot be solved in polynomial time. It also partly explains why we are interested in proving that problems are NP-complete.

Theorem

If A is NP-complete and $A \leq_P B$ and $B \in NP$ then B is NP-complete.

Theorem

If A is NP-complete and $A \in P$, then $P = NP$.

The theorem serves as an indication that NP-complete problems probably cannot be solved in polynomial time. It also partly explains why we are interested in proving that problems are NP-complete.

Theorem

If A is NP-complete and $A \leq_P B$ and $B \in NP$ then B is NP-complete.

Proof

We already know that $B \in NP$. We need to show that all languages in NP can be reduced to B. Since we know that any language in NP can be reduced to A, and that A can be reduced to B, we can reduce any language in NP to B. Here we use the fact that polynomial reductions can be composed to create new polynomial reductions. $\square$

The "first" *NP*-complete problem

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}.$

The "first" *NP*-complete problem

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$.
We will be working with Boolean formulas on a special form called *conjunctive normal form* (CNF). A formula on CNF consists of several *clauses* joined by conjunctions ($\wedge$) like this:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k.$$

The "first" *NP*-complete problem

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$.
We will be working with Boolean formulas on a special form called *conjunctive normal form* (CNF). A formula on CNF consists of several *clauses* joined by conjunctions ($\wedge$) like this:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k.$$

Each clause consists of several *literals* joined by disjunctions ($\vee$). Literals are Boolean variables or negated Boolean variables (x or $\overline{x}$). An example of a formula on CNF could be:

$$\phi = (x \vee \overline{x}) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z} \vee z).$$

The "first" *NP*-complete problem

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$.
We will be working with Boolean formulas on a special form called *conjunctive normal form* (CNF). A formula on CNF consists of several *clauses* joined by conjunctions ($\wedge$) like this:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k.$$

Each clause consists of several *literals* joined by disjunctions ($\vee$). Literals are Boolean variables or negated Boolean variables (x or $\overline{x}$). An example of a formula on CNF could be:

$$\phi = (x \vee \overline{x}) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{z} \vee z).$$

$CNFSAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula, on CNF}\}$.

The "first" *NP*-complete problem II

Theorem (Cook-Levin)

SAT is NP-complete.

The "first" NP-complete problem II

Theorem (Cook-Levin)

SAT is NP-complete.

We will not go through the proof, but we will try to get a grip on the fundamental parts.
Proof overview

- Show that $SAT \in NP$ (last lecture)
- Create a universal reduction from $A \in NP$ to SAT
- The reduction takes an input of A, let us call it w, and produces a formula ϕ
  - Since $A \in NP$, there exists a NTM M_A deciding A in time n^k for some constant k
  - Create a formula ϕ such that ϕ is satisfiable if and only if M_A has an accepting branch in its computation on input w

The "first" *NP*-complete problem III

The formula will "simulate" M_A on input w. Here is a (simplified) draft of ϕ:

$$\phi = \mathit{Init} \wedge \mathit{Legal} \wedge \mathit{Accepting}.$$

The "first" NP-complete problem III

The formula will "simulate" M_A on input w. Here is a (simplified) draft of ϕ:

$$\phi = \mathit{Init} \wedge \mathit{Legal} \wedge \mathit{Accepting}.$$

An essential part of the proof is to show that the reduction only takes polynomial time in the length of w. In a complete proof we would have to carefully analyze each step of creating ϕ. Furthermore, it is possible to create ϕ to be on CNF, which actually proves that *CNFSAT* is NP-complete. This will come in handy later.

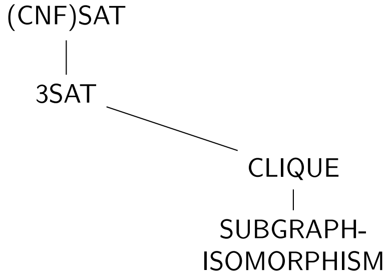
Map of *NP*-complete problems

(CNF)SAT

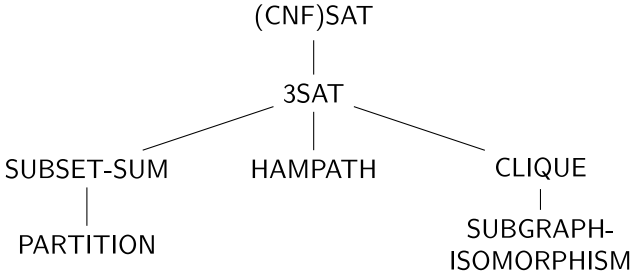
|

3SAT

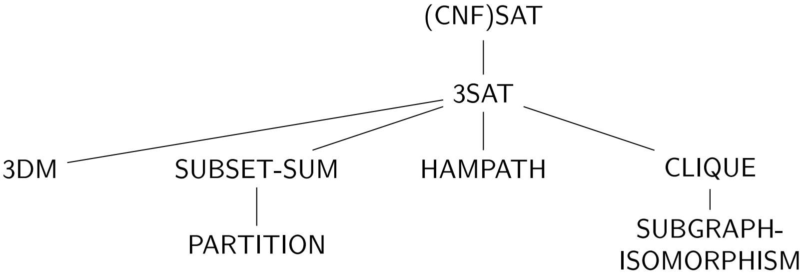
Map of *NP*-complete problems



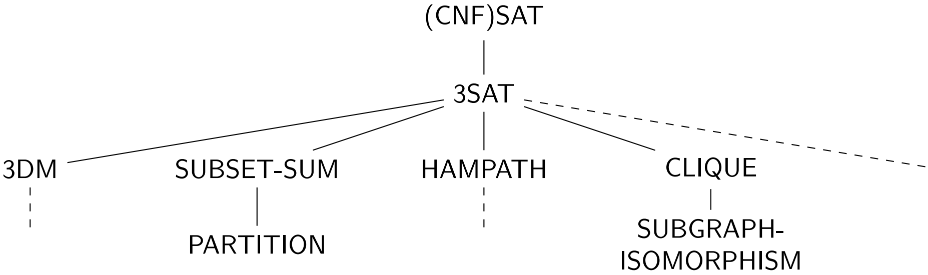
Map of *NP*-complete problems



Map of *NP*-complete problems



Map of *NP*-complete problems



Actually, since all problems in NP can be (poly-time) reduced to any NP-complete problem, and all NP-complete problems are in NP, all NP-complete problems can be reduced to each other in polynomial time.

We say that a Boolean formula is on 3CNF if it is on CNF and each clause has three literals.
Let $3SAT = \{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean formula, on 3CNF$\}$.

We say that a Boolean formula is on 3CNF if it is on CNF and each clause has three literals.
Let $3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula, on 3CNF}\}$.

Theorem

$3SAT$ is NP-complete.

We say that a Boolean formula is on 3CNF if it is on CNF and each clause has three literals.
Let $3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula, on 3CNF}\}$.

Theorem

$3SAT$ is NP-complete.

Proof that $3SAT$ is NP-complete: Part I

First we need to show that $3SAT$ is in NP. Here we can still use a satisfying assignment as our certificate.
To show that all problems in NP can be polynomial time reduced to $3SAT$ it is enough to show that $CNFSAT \leq_P 3SAT$. Such a reduction will take a formula ϕ on CNF and output a formula ψ on 3CNF, such that ϕ is satisfiable if and only if ψ is satisfiable. We show how to do this on the next slide. $\square$

Proof that $3SAT$ is NP-complete: Part II

For each clause of ϕ, we replace it with corresponding clauses having exactly 3 literals. If a clause has one or two literals in ϕ, we pad these clauses with repeated literals so that we end up with three literals.

Proof that $3SAT$ is NP-complete: Part II

For each clause of ϕ, we replace it with corresponding clauses having exactly 3 literals. If a clause has one or two literals in ϕ, we pad these clauses with repeated literals so that we end up with three literals. For example

$$(x_1 \vee \overline{x_2})$$

is transformed into

$$(x_1 \vee \overline{x_2} \vee x_1).$$

Proof that $3SAT$ is NP-complete: Part II

For each clause of ϕ, we replace it with corresponding clauses having exactly 3 literals. If a clause has one or two literals in ϕ, we pad these clauses with repeated literals so that we end up with three literals. For example

$$(x_1 \vee \overline{x_2})$$

is transformed into

$$(x_1 \vee \overline{x_2} \vee x_1).$$

If a clause has more than three literals, we split it up into several new clauses and link them together with new literals. For example

$$(x_1 \vee \overline{x_2} \vee x_3 \vee x_4 \vee \overline{x_5})$$

is replaced by the clauses

$$(x_1 \vee \overline{x_2} \vee z_1), (\overline{z_1} \vee x_3 \vee z_2), (\overline{z_2} \vee x_4 \vee \overline{x_5}).$$

Proof that $3SAT$ is NP-complete: Part II

For each clause of ϕ, we replace it with corresponding clauses having exactly 3 literals. If a clause has one or two literals in ϕ, we pad these clauses with repeated literals so that we end up with three literals. For example

$$(x_1 \vee \overline{x_2})$$

is transformed into

$$(x_1 \vee \overline{x_2} \vee x_1).$$

If a clause has more than three literals, we split it up into several new clauses and link them together with new literals. For example

$$(x_1 \vee \overline{x_2} \vee x_3 \vee x_4 \vee \overline{x_5})$$

is replaced by the clauses

$$(x_1 \vee \overline{x_2} \vee z_1),\ (\overline{z_1} \vee x_3 \vee z_2),\ (\overline{z_2} \vee x_4 \vee \overline{x_5}).$$

Here the z_i are fresh variables not mentioned in ϕ and they work as "logical glue" in the reduction. Finally our reduction outputs ψ, the conjunction of our new clauses. It has polynomial size compared to ϕ, and preserves satisfiability. Since this reduction shows that $CNFSAT \leq_P 3SAT$ we have proven that $3SAT$ is NP-complete. $\square$

Recall the problem *CLIQUE* from last lecture:
$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a clique of size } k\}$

Recall the problem *CLIQUE* from last lecture:
$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a clique of size } k\}$

Theorem

CLIQUE is NP-complete.

Recall the problem *CLIQUE* from last lecture:
$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a clique of size } k\}$

Theorem

CLIQUE is NP-complete.

Proof that *CLIQUE* is *NP*-complete: Part I

Last lecture we discussed possible polynomial certificates for *CLIQUE*, so we conclude that $CLIQUE \in NP$. To show that *CLIQUE* is *NP*-hard, we show that $3SAT \leq_P CLIQUE$. The reduction f takes a formula $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_k$ on 3CNF, and generates the string $\langle G, k\rangle$, such that ϕ is satisfiable iff G has a clique of size k. We will show that f works correctly, and that it is computable in polynomial time. □

Proof that *CLIQUE* is *NP*-complete: Part II

The graph G will consist of k groups of three nodes called *triples*. Each triple will correspond to a clause in ϕ and each node in a triple will correspond to a literal in the corresponding clause of ϕ. We can label the nodes with its corresponding literal in ϕ.

Proof that *CLIQUE* is *NP*-complete: Part II

The graph G will consist of k groups of three nodes called *triples*. Each triple will correspond to a clause in ϕ and each node in a triple will correspond to a literal in the corresponding clause of ϕ. We can label the nodes with its corresponding literal in ϕ. Then we connect all pairs of nodes except:

1. pairs of nodes that are in the same triple
2. pairs of nodes where one is labeled x and the other $\overline{x}$ for some variable x.

Proof that *CLIQUE* is *NP*-complete: Part II

The graph G will consist of k groups of three nodes called *triples*. Each triple will correspond to a clause in ϕ and each node in a triple will correspond to a literal in the corresponding clause of ϕ. We can label the nodes with its corresponding literal in ϕ. Then we connect all pairs of nodes except:

1. pairs of nodes that are in the same triple
2. pairs of nodes where one is labeled x and the other $\overline{x}$ for some variable x.

Now we claim that G has a clique of size k if and only if ϕ can be satisfied.

Proof that *CLIQUE* is *NP*-complete: Part II

The graph G will consist of k groups of three nodes called *triples*. Each triple will correspond to a clause in ϕ and each node in a triple will correspond to a literal in the corresponding clause of ϕ. We can label the nodes with its corresponding literal in ϕ. Then we connect all pairs of nodes except:

1. pairs of nodes that are in the same triple
2. pairs of nodes where one is labeled x and the other $\overline{x}$ for some variable x.

Now we claim that G has a clique of size k if and only if ϕ can be satisfied.
If G has a k-clique, then since no two nodes in a triple is connected, the clique must contain exactly one node from each of the k triples. We assign truth values to the variables by making the literals of the nodes in the clique true. This is always possible since no contradictory nodes are not connected. For example, if $\overline{x}$ is the label of a node in the clique, then no node labeled x can be in the clique, so it is "safe" to set x to false. Assigning the variables like described will satisfy ϕ since one literal in each clause is true. Thus, in this case ϕ is satisfiable.

$\square$

Proof that *CLIQUE* is *NP*-complete: Part III

Assume ϕ is satisfiable. Then there exists an assignment making at least on literal true in each clause. We form a k-clique in G by selecting a node corresponding to a true literal from each triple. Since there are k triples in G the size of the clique is k. Furthermore, all the nodes will have an edge between them since they are from different triples and two nodes representing contradictory literals could have been selected.

Proof that *CLIQUE* is *NP*-complete: Part III

Assume ϕ is satisfiable. Then there exists an assignment making at least on literal true in each clause. We form a k-clique in G by selecting a node corresponding to a true literal from each triple. Since there are k triples in G the size of the clique is k. Furthermore, all the nodes will have an edge between them since they are from different triples and two nodes representing contradictory literals could have been selected.
Finally, we claim that the reduction runs in polynomial time, since the number of nodes in G equals the number of literals in ϕ (linear), and the number of edges is no more than n^2 (quadratic).

Proof that *CLIQUE* is *NP*-complete: Part III

Assume ϕ is satisfiable. Then there exists an assignment making at least on literal true in each clause. We form a k-clique in G by selecting a node corresponding to a true literal from each triple. Since there are k triples in G the size of the clique is k. Furthermore, all the nodes will have an edge between them since they are from different triples and two nodes representing contradictory literals could have been selected.

Finally, we claim that the reduction runs in polynomial time, since the number of nodes in G equals the number of literals in ϕ (linear), and the number of edges is no more than n^2 (quadratic).

Since we have shown both that $3SAT \leq_P CLIQUE$ and $CLIQUE \in NP$, we have proven the theorem. $\square$

Let G and H be two graphs. We say that G is isomorphic to H if there exists a bijection f from the set of nodes in G to the set of nodes in H, such that u and v are neighbors in G if and only if $f(u)$ and $f(v)$ are neighbors in H.
Let $SUBGRAPH\text{–}ISOMORPHISM = \{\langle G_1, G_2 \rangle \mid G_1 \text{ is isomorphic to a subgraph of } G_2\}$.

Let G and H be two graphs. We say that G is isomorphic to H if there exists a bijection f from the set of nodes in G to the set of nodes in H, such that u and v are neighbors in G if and only if $f(u)$ and $f(v)$ are neighbors in H.
Let $SUBGRAPH\text{–}ISOMORPHISM = \{\langle G_1, G_2\rangle \mid G_1 \text{ is isomorphic to a subgraph of } G_2\}$.

Theorem

SUBGRAPH–ISOMORPHISM is NP-complete.

Let G and H be two graphs. We say that G is isomorphic to H if there exists a bijection f from the set of nodes in G to the set of nodes in H, such that u and v are neighbors in G if and only if $f(u)$ and $f(v)$ are neighbors in H.
Let $SUBGRAPH\text{–}ISOMORPHISM = \{\langle G_1, G_2\rangle \mid G_1 \text{ is isomorphic to a subgraph of } G_2\}$.

Theorem

SUBGRAPH–ISOMORPHISM is NP-complete.

Proof

To show that the problem is in NP, we claim that a bijection from G_1 to a subgraph of G_2 is a suitable polynomial certificate. Our verifier could then check the neighbors and confirm that G_1 is isomorphic to a subgraph of G_2.

Let G and H be two graphs. We say that G is isomorphic to H if there exists a bijection f from the set of nodes in G to the set of nodes in H, such that u and v are neighbors in G if and only if $f(u)$ and $f(v)$ are neighbors in H.
Let $SUBGRAPH\text{–}ISOMORPHISM = \{\langle G_1, G_2 \rangle \mid G_1 \text{ is isomorphic to a subgraph of } G_2\}$.

Theorem

SUBGRAPH–ISOMORPHISM is NP-complete.

Proof

To show that the problem is in NP, we claim that a bijection from G_1 to a subgraph of G_2 is a suitable polynomial certificate. Our verifier could then check the neighbors and confirm that G_1 is isomorphic to a subgraph of G_2.
To show that $SUBGRAPH\text{–}ISOMORPHISM$ is NP-hard, we will show that $CLIQUE \leq_P SUBGRAPH\text{–}ISOMORPHISM$. The reduction will take a string $\langle G, k \rangle$ as input and it will produce $\langle G_1, G_2 \rangle$ such that G_1 is isomorphic to a subgraph of G_2 iff G contains a k-clique. We let G_1 be a k-clique and $G_2 = G$. The correctness is trivial. Furthermore, the reduction runs in polynomial time; it copies G (linear) and creates a k-clique (can be done in quadratic time with respect to k). $\square$

A template for proving NP-completeness

Let $MY\text{–}PROB = \{\ w \mid w$ has some property $R\}$. Prove that $MY\text{–}PROB$ is NP-complete.

A template for proving NP-completeness

Let $MY\text{–}PROB = \{\ w \mid w \text{ has some property } R\}$. Prove that $MY\text{–}PROB$ is NP-complete.

Proof

Show $MY\text{–}PROB \in NP$. This is typically the easiest part of the proof. Try to think of a certificate that w has property R. Remember that a verifier only has polynomial time, so it cannot hope to read exponential certificates.

A template for proving NP-completeness

Let $MY{-}PROB = \{\ w \mid w \text{ has some property } R\}$. Prove that $MY{-}PROB$ is NP-complete.

Proof

Show $MY{-}PROB \in NP$. This is typically the easiest part of the proof. Try to think of a certificate that w has property R. Remember that a verifier only has polynomial time, so it cannot hope to read exponential certificates.

Now we show that $MY{-}PROB$ is NP-hard. To do this, it is sufficient to prove that a known NP-complete problem can be reduced to $MY{-}PROB$. Pick the the NP-complete problem A, that resembles $MY{-}PROB$ the most. This often makes the reduction simpler. Then, we need to show that $A \leq_P MY{-}PROB$. To do this, we create a polynomial time reduction, mapping w_A (instances of A) to w (instances of $MY{-}PROB$) such that w_A has property R_A if and only if w has property R. This is the part of the proof that may involve some ingenuity on our side. Try to understand how the two problems can be related.

A template for proving NP-completeness

Let $MY{-}PROB = \{ \ w \mid w$ has some property $R\}$. Prove that $MY{-}PROB$ is NP-complete.

Proof

Show $MY{-}PROB \in NP$. This is typically the easiest part of the proof. Try to think of a certificate that w has property R. Remember that a verifier only has polynomial time, so it cannot hope to read exponential certificates.

Now we show that $MY{-}PROB$ is NP-hard. To do this, it is sufficient to prove that a known NP-complete problem can be reduced to $MY{-}PROB$. Pick the the NP-complete problem A, that resembles $MY{-}PROB$ the most. This often makes the reduction simpler. Then, we need to show that $A \leq_P MY{-}PROB$. To do this, we create a polynomial time reduction, mapping w_A (instances of A) to w (instances of $MY{-}PROB$) such that w_A has property R_A if and only if w has property R. This is the part of the proof that may involve some ingenuity on our side. Try to understand how the two problems can be related.

After explaining how the reduction works, we will typically argue that it is correct. First assume $w_A \in A$ and show that $w \in MY{-}PROB$. Then, either assume $w \in MY{-}PROB$ and show that $w_A \in A$, or assume $w_A \notin A$ and show that $w \notin MY{-}PROB$.

A template for proving NP-completeness

Let $MY\text{−}PROB = \{\ w \mid w$ has some property $R\}$. Prove that $MY\text{−}PROB$ is NP-complete.

Proof

Show $MY\text{−}PROB \in NP$. This is typically the easiest part of the proof. Try to think of a certificate that w has property R. Remember that a verifier only has polynomial time, so it cannot hope to read exponential certificates.

Now we show that $MY\text{−}PROB$ is NP-hard. To do this, it is sufficient to prove that a known NP-complete problem can be reduced to $MY\text{−}PROB$. Pick the the NP-complete problem A, that resembles $MY\text{−}PROB$ the most. This often makes the reduction simpler. Then, we need to show that $A \leq_P MY\text{−}PROB$. To do this, we create a polynomial time reduction, mapping w_A (instances of A) to w (instances of $MY\text{−}PROB$) such that w_A has property R_A if and only if w has property R. This is the part of the proof that may involve some ingenuity on our side. Try to understand how the two problems can be related.

After explaining how the reduction works, we will typically argue that it is correct. First assume $w_A \in A$ and show that $w \in MY\text{−}PROB$. Then, either assume $w \in MY\text{−}PROB$ and show that $w_A \in A$, or assume $w_A \notin A$ and show that $w \notin MY\text{−}PROB$.

Finally we explain that the reduction can be carried out in polynomial time with respect to w_A. How hard this is, will depend on the reduction. Often, just a few lines is sufficient.

A template for proving NP-completeness

Let $MY{-}PROB = \{\ w \mid w \text{ has some property } R\}$. Prove that $MY{-}PROB$ is NP-complete.

Proof

Show $MY{-}PROB \in NP$. This is typically the easiest part of the proof. Try to think of a certificate that w has property R. Remember that a verifier only has polynomial time, so it cannot hope to read exponential certificates.

Now we show that $MY{-}PROB$ is NP-hard. To do this, it is sufficient to prove that a known NP-complete problem can be reduced to $MY{-}PROB$. Pick the the NP-complete problem A, that resembles $MY{-}PROB$ the most. This often makes the reduction simpler. Then, we need to show that $A \leq_P MY{-}PROB$. To do this, we create a polynomial time reduction, mapping w_A (instances of A) to w (instances of $MY{-}PROB$) such that w_A has property R_A if and only if w has property R. This is the part of the proof that may involve some ingenuity on our side. Try to understand how the two problems can be related.

After explaining how the reduction works, we will typically argue that it is correct. First assume $w_A \in A$ and show that $w \in MY{-}PROB$. Then, either assume $w \in MY{-}PROB$ and show that $w_A \in A$, or assume $w_A \notin A$ and show that $w \notin MY{-}PROB$.

Finally we explain that the reduction can be carried out in polynomial time with respect to w_A. How hard this is, will depend on the reduction. Often, just a few lines is sufficient.

We end the proof by explaining that we have showed both that $MY{-}PROB \in NP$ and that $MY{-}PROB$ is NP-hard, so therefore $MY{-}PROB$ is NP-complete. $\square$

Let $SUBSET\text{–}SUM = \{\langle S, t\rangle \ \mid S$ is a multiset of **natural numbers**, such that there exists a subset of S summing to $t\}$. $SUBSET\text{–}SUM$ can be shown NP-complete by a reduction from $3SAT$.

Let $SUBSET\text{--}SUM = \{\langle S, t\rangle \ \mid S$ is a multiset of **natural numbers**, such that there exists a subset of S summing to $t\}$. $SUBSET\text{--}SUM$ can be shown NP-complete by a reduction from $3SAT$.

Let $PARTITION = \{\langle S\rangle \mid S$ is a multiset of natural numbers, such that there exists a subset S' of S that sums to exactly half the sum of $S\}$.

Theorem

PARTITION is NP-complete.

Let $SUBSET\text{–}SUM = \{\langle S, t \rangle \ \mid S$ is a multiset of **natural numbers**, such that there exists a subset of S summing to $t\}$. $SUBSET\text{–}SUM$ can be shown NP-complete by a reduction from $3SAT$.

Let $PARTITION = \{\langle S \rangle \mid S$ is a multiset of natural numbers, such that there exists a subset S' of S that sums to exactly half the sum of $S\}$.

Theorem

$PARTITION$ is NP-complete.

Proof that $PARTITION$ is NP-complete: Part I

To show that $PARTITION \in NP$, we need to find short certificates for yes-instances of $PARTITION$. Such a certificate could be a subset of S summing to half the sum of S.

Let $SUBSET\text{--}SUM = \{\langle S, t\rangle \mid S$ is a multiset of **natural numbers**, such that there exists a subset of S summing to $t\}$. $SUBSET\text{--}SUM$ can be shown NP-complete by a reduction from $3SAT$.

Let $PARTITION = \{\langle S\rangle \mid S$ is a multiset of natural numbers, such that there exists a subset S' of S that sums to exactly half the sum of $S\}$.

Theorem

PARTITION is NP-complete.

Proof that $PARTITION$ is NP-complete: Part I

To show that $PARTITION \in NP$, we need to find short certificates for yes-instances of $PARTITION$. Such a certificate could be a subset of S summing to half the sum of S.

We will show that $SUBSET\text{--}SUM \leq_P PARTITION$. The reduction is given $\langle S, t\rangle$, where $\Sigma S = k$. If $t > k$, we already know that we are dealing with a no-instance, so we return the set $\{1\}$, which we know is a no-instance of $PARTITION$. We then construct the set S' to be $S \cup \{N_1, N_2\}$, where $N_1 = 2k - t$ and $N_2 = k + t$. The sum of S' is $k + 2k - t + k + t = 4k$. Since $N_1 + N_2$ is more than half the sum of S' they must end up in different subsets in a correct partition of S'. □

Proof that *PARTITION* is *NP*-complete: Part II

Assume $\langle S, t \rangle \in SUBSET\text{–}SUM$. Then there exists a subset of S, let us call it S_1, summing to t. This implies that $S \setminus S_1$ has a sum of $k - t$. Then, $\langle S' \rangle \in PARTITION$ since $\{N_1\} \cup S_1$ has a sum of $2k$ which is the same as the sum of $\{N_2\} \cup (S \setminus S_1)$.

Proof that *PARTITION* is *NP*-complete: Part II

Assume $\langle S, t \rangle \in \mathit{SUBSET{-}SUM}$. Then there exists a subset of S, let us call it S_1, summing to t. This implies that $S \setminus S_1$ has a sum of $k - t$. Then, $\langle S' \rangle \in \mathit{PARTITION}$ since $\{N_1\} \cup S_1$ has a sum of $2k$ which is the same as the sum of $\{N_2\} \cup (S \setminus S_1)$.

Assume $\langle S' \rangle \in \mathit{PARTITION}$. Then, since we know that N_1 and N_2 has to be in different subsets, $\langle S, t \rangle \in \mathit{SUBSET{-}SUM}$.

Proof that *PARTITION* is *NP*-complete: Part II

Assume $\langle S, t\rangle \in SUBSET\text{–}SUM$. Then there exists a subset of S, let us call it S_1, summing to t. This implies that $S \setminus S_1$ has a sum of $k - t$. Then, $\langle S'\rangle \in PARTITION$ since $\{N_1\} \cup S_1$ has a sum of $2k$ which is the same as the sum of $\{N_2\} \cup (S \setminus S_1)$.
Assume $\langle S'\rangle \in PARTITION$. Then, since we know that N_1 and N_2 has to be in different subsets, $\langle S, t\rangle \in SUBSET\text{–}SUM$.
The reduction is computable in polynomial time, since we added only two elements to S, which both are at most twice the size of the sum of S. Furthermore, computing k can be done in polynomial time.

Proof that *PARTITION* is *NP*-complete: Part II

Assume $\langle S, t \rangle \in SUBSET\text{–}SUM$. Then there exists a subset of S, let us call it S_1, summing to t. This implies that $S \setminus S_1$ has a sum of $k - t$. Then, $\langle S' \rangle \in PARTITION$ since $\{N_1\} \cup S_1$ has a sum of $2k$ which is the same as the sum of $\{N_2\} \cup (S \setminus S_1)$.
Assume $\langle S' \rangle \in PARTITION$. Then, since we know that N_1 and N_2 has to be in different subsets, $\langle S, t \rangle \in SUBSET\text{–}SUM$.
The reduction is computable in polynomial time, since we added only two elements to S, which both are at most twice the size of the sum of S. Furthermore, computing k can be done in polynomial time.
We conclude that *PARTITION* is *NP*-complete. $\square$