Introduction to Algorithms and Data Structures

Vertex Cover and Other NP-complete problems
Polynomial Time Reduction

• We are given a problem \( A \) that we want to solve.

• We can reduce solving problem \( A \) to solving some other problem \( B \).

• Assume that we had an algorithm \( \text{ALG}^B \) for solving problem \( B \), which we can use at cost \( O(1) \).

• We can construct an algorithm \( \text{ALG}^A \) for solving problem \( A \), which uses calls to the algorithm \( \text{ALG}^B \) as a subroutine.

• If \( \text{ALG}^A \) is a polynomial time algorithm, then this is a polynomial time reduction.
Pictorially

Problem A

Do stuff …
Do stuff …
Do stuff …

ALG^A

instance transformation

Problem B

ALG^B
Types of reductions

• **Turing reduction:**

  • Argument: Here is an algorithm which runs in polynomial time solving problem A, using polynomially many calls to an oracle for problem B.

• **Many-one reduction:**

  • Argument:

    • If \( z \) is a solution to instance \( I \) of problem A, then \( z' \) is a solution of instance \( f(I) \) to problem B.

    • If \( z \) is not a solution to instance \( I \) of problem A, then \( z' \) is not a solution of instance \( f(I) \) to problem B.

    • Equivalently: If \( z' \) is a solution of instance \( f(I) \) to problem B, then \( z \) is a solution to instance \( I \) of problem A.
How to work with reductions

- **Positive:** Assume that I want to solve problem A and I know how to solve problem B in polynomial time.

  - I can try to come up with a polynomial time reduction $A \leq_p B$, which will give me a polynomial time algorithm for solving A.

- **Contrapositive:** Assume that there is a problem A for which it is unlikely that there is a polynomial time algorithm that solves it.

  - If I come up with a polynomial time reduction $A \leq_p B$, it is also unlikely that there is a polynomial time algorithm that solves B.

  - B is “at least as hard to solve as” A, because if I could solve B, I could also solve A.
How to work with reductions

- **Positive:** Assume that I want to solve problem A and I know how to solve problem B in polynomial time.
  - I can try to come up with a polynomial time reduction $A \leq_p B$, which will give me a polynomial time algorithm for solving A.

- **Contrapositive:** Assume that there is a problem A for which it is unlikely that there is a polynomial time algorithm that solves it.
  - If I come up with a polynomial time reduction $A \leq_p B$, it is also unlikely that there is a polynomial time algorithm that solves B.
  - B is “at least as hard to solve as” A, because if I could solve B, I could also solve A.
3 SAT

• A CNF formula with $m$ clauses and $k$ literals.

$$\phi = (x_1 \lor x_5 \lor x_3) \land (x_2 \lor x_6 \lor \neg x_5) \land \ldots \land (x_3 \lor x_8 \lor x_12)$$

• (“An AND of ORs”).

• Each clause has three literals.
3 SAT

- A CNF formula with $m$ clauses and $k$ literals.

\[ \phi = (x_1 \lor x_5 \lor x_3) \land (x_2 \lor x_6 \lor \overline{x_5}) \land ... \land (x_3 \lor x_8 \lor x_{12}) \]

- (“An AND of ORs”).

- Each clause has three literals.

- Truth assignment: A value in \{0,1\} for each variable $x_i$. 
3 SAT

• A CNF formula with $m$ clauses and $k$ literals.

$$\phi = (x_1 \lor x_5 \lor x_3) \land (x_2 \lor x_6 \lor \neg x_5) \land \ldots \land (x_3 \lor x_8 \lor x_{12})$$

• (“An AND of ORs”).

• Each clause has three literals.

• Truth assignment: A value in \{0,1\} for each variable $x_i$.

• Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).
3 SAT

- A CNF formula with \( m \) clauses and \( k \) literals.

\[
\phi = (x_1 \lor x_5 \lor x_3) \land (x_2 \lor x_6 \lor \neg x_5) \land \ldots \land (x_3 \lor x_8 \lor x_{12})
\]

- (“An AND of ORs”).

- Each clause has three literals.

- Truth assignment: A value in \{0,1\} for each variable \( x_i \).

- Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).

- Computational problem 3SAT: Decide if the input formula \( \phi \) has a satisfying assignment.
3 SAT is NP-complete
3 SAT is NP-complete

- 3 SAT is in NP
3 SAT is NP-complete

- 3 SAT is in NP
- 3 SAT is NP-hard.
3 SAT is NP-complete

- 3 SAT is in NP
- 3 SAT is NP-hard.

Remarks:

- The first problem shown to be NP-complete was the SAT problem (more general than 3 SAT), and this reduces to 3SAT.
- Several textbooks start from Circuit SAT, a version of the SAT problem defined on circuits with boolean gates AND, OR or NOT.
Proving NP-completeness
Proving NP-completeness

• Suppose that you are given a problem $A$ and you want to prove that it is NP-complete.
Proving NP-completeness

• Suppose that you are given a problem A and you want to prove that it is NP-complete.

• First, prove that A is in NP.
  • Usually by observing that a solution is efficiently checkable.
Proving NP-completeness

- Suppose that you are given a problem A and you want to prove that it is NP-complete.

- First, prove that A is in NP.
  - Usually by observing that a solution is efficiently checkable.

- Then prove that A is NP-hard.
  - Construct a polynomial time reduction from some NP-complete (or just NP-hard) problem P.
Enough with the definitions. Let’s see how it works.

• We will prove that a well-known problem on graphs, called **Vertex Cover** is **NP-complete**.
Definition: A vertex cover $C$ of a graph $G=(V, E)$ is a subset of the nodes such that every edge $e$ in the graph has at least one endpoint in $C$.

Definition: A minimum vertex cover is a vertex cover of the smallest possible size.

Vertex Cover
Input: A graph $G=(V, E)$
Output: A minimum vertex cover.
Example
Example
Example
Example

A vertex cover
Example
Example
Example
Example

A minimum vertex cover
Vertex Cover

- **Definition:** A vertex cover \( C \) of a graph \( G=(V, E) \) is a subset of the nodes such that every edge \( e \) in the graph has at least one endpoint in \( C \).

- **Definition:** A minimum vertex cover is a vertex cover of the smallest possible size.

- **Vertex Cover**
  - **Input:** A graph \( G=(V, E) \)
  - **Output:** A minimum vertex cover.
Definition: A vertex cover $C$ of a graph $G=(V, E)$ is a subset of the nodes such that every edge $e$ in the graph has at least one endpoint in $C$.

Definition: A minimum vertex cover is a vertex cover of the smallest possible size.

Vertex Cover
Input: A graph $G=(V, E)$ and a number $k$
Output: Is there a vertex cover of size $\leq k$?.
Vertex cover
Vertex cover

- Vertex Cover is in \textit{NP}.
Vertex cover

- Vertex Cover is in $\text{NP}$. 

- Assume that we are given a vertex cover.

  - We can check that is has size $k$ and that it is a vertex cover in polynomial time.
Vertex cover

- Vertex Cover is in NP-hard.
Vertex cover

- Vertex Cover is in **NP-hard**.

- We will construct a polynomial time reduction from 3SAT.
  - i.e., we will prove that $3SAT \leq^p \text{Vertex Cover.}$
The reduction

- Let $\phi$ be a 3-CNF formula with $m$ clauses and $d$ variables.

- We construct, in polynomial time, an instance $<G, k>$ of Vertex Cover such that

  - If $\phi$ is satisfiable $\Rightarrow$ $G$ has a vertex cover of size at most $k$.

  - If $\phi$ is not satisfiable $\Rightarrow$ $G$ does not have any vertex cover of size at most $k$. 
The reduction

- For every variable $x$ in $\phi$, we create two nodes $x$ and $\neg x$ in $G$ and we connect them with an edge $e = (x, \neg x)$.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
The reduction

• For every variable $x$ in $\phi$, we create two nodes $x$ and $\overline{x}$ in $G$ and we connect them with an edge $e = (x, \overline{x})$.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_2 \lor x_2)$
The reduction

For every clause \( l = (l_1, l_2, l_3) \) in \( \phi \), we create three nodes \( l_1, l_2, l_3 \) in \( G \) and we connect them all with each other.

Running example: \( \phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2) \)
The reduction

• We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
The reduction

- We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
The reduction

- We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
The reduction

- We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
The reduction

• We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
The reduction

- We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \lor (\neg x_1 \lor x_2 \lor x_2)$
The reduction

- We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
The reduction

• We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\lnot x_1 \lor \lnot x_2 \lor \lnot x_2) \land (\lnot x_1 \lor x_2 \lor x_2)$
The reduction

- We add an edge between all nodes with the same label on the top and on the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
The reduction

- We add an edge between all nodes with the same label on the top and on the bottom.

Running example: \( \phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2) \)
The reduction

- Let $\phi$ be a 3-CNF formula with $m$ clauses and $d$ variables.
- We construct, in polynomial time, an instance $<G, k>$ of Vertex Cover, with $k = d + 2m$ such that
  - If $\phi$ is satisfiable $\implies$ $G$ has a vertex cover of size at most $k$.
  - If $\phi$ is not satisfiable $\implies$ $G$ does not have any vertex cover of size at most $k.$
One direction
One direction

- If $\phi$ is satisfiable $\Rightarrow$ G has a vertex cover of size at most k.
One direction

• If $\phi$ is satisfiable $\Rightarrow$ $G$ has a vertex cover of size at most $k$.

• Let $(y_1, y_2, \ldots, y_k)$ in $\{0,1\}^n$ be a satisfying assignment for $\phi$. 
One direction

- If $\phi$ is satisfiable $\Rightarrow$ $G$ has a vertex cover of size at most $k$.

- Let $(y_1, y_2, \ldots, y_k)$ in $\{0,1\}^n$ be a satisfying assignment for $\phi$.

- For the nodes on the top: If $y_i = 1$, include node $x_i$ in the vertex cover $C$, otherwise, include node $\neg x_i$. 
One direction

• If $\phi$ is satisfiable $\Rightarrow$ $G$ has a vertex cover of size at most $k$.

• Let $(y_1, y_2, \ldots, y_k)$ in $\{0, 1\}^n$ be a satisfying assignment for $\phi$.

• For the nodes on the top: If $y_i = 1$, include node $x_i$ in the vertex cover $C$, otherwise, include node $\neg x_i$.

• For the nodes on the bottom: In each triangle, choose a node $x_i$ that has been picked on the top and do not include it in the vertex cover. Include the other two nodes.
Example

- For the nodes on the top: If $y_i = 1$, include node $x_i$ in the vertex cover $C$, otherwise, include node $\neg x_i$.

- Assume $y_1 = 0$, $y_2 = 1$.

Running example: $\phi = (x_1 \lor \neg x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
Example

- For the nodes on the top: If \( y_i = 1 \), include node \( x_i \) in the vertex cover \( C \), otherwise, include node \( \overline{x_i} \).

- Assume \( y_1 = 0 \), \( y_2 = 1 \).

Running example: \( \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_2 \lor x_2) \)
Example

- For the nodes on the bottom: In each triangle, choose a note $x_i$ that has been picked on the top and do not include it in the vertex cover. Include the other two nodes.

- Assume $y_1 = 0$, $y_2 = 1$.

Running example: $\phi = (x_1 \lor \neg x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
Example

• For the nodes on the bottom: In each triangle, choose a note $x_i$ that has been picked on the top and do not include it in the vertex cover. Include the other two nodes.

• Assume $y_1 = 0$, $y_2 = 1$.

Running example: $\phi = (x_1 \vee \neg x_1 \vee x_2) \land (\neg x_1 \vee \neg x_2 \vee \neg x_2) \land (\neg x_1 \vee x_2 \vee x_2)$
One direction

• **Claim:** The set of nodes we have chosen is a vertex cover.

• Every edge on the top is incident to either node $x_i$ or node $\overline{x}_i$.

• Every edge on the bottom is incident to some node in the set, since we select two out of three nodes.

• Every edge between the top and to bottom is incident to some node.
Example

• For the nodes on the bottom: In each triangle, choose a note $x_i$ that has been picked on the top and do not include it in the vertex cover. Include the other two nodes.

• Assume $y_1 = 0$, $y_2 = 1$.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
One direction

• **Claim:** The vertex cover has size $k = d + 2m$

  • Each variable is selected at the top (either as $x_i$ or as $\neg x_i$).

  • For each clause, we select two nodes at the bottom.
Other direction

• If \( \phi \) is not satisfiable \( \Rightarrow \) \( G \) does not have any vertex cover of size at most \( k \).
Other direction

- If $\phi$ is not satisfiable $\Rightarrow$ $G$ does not have any vertex cover of size at most $k$.

- $G$ has a vertex cover of size at most $k$. $\Rightarrow$ $\phi$ is satisfiable.
Other direction

- $G$ has a vertex cover of size at most $k$. $\implies \phi$ is satisfiable.
Other direction

• G has a vertex cover of size at most $k$. $\Rightarrow \phi$ is satisfiable.

• Let $C$ be a vertex cover of size $k = d + 2m$ in $G$. 
Other direction

• G has a vertex cover of size at most $k$. $\Rightarrow \phi$ is satisfiable.

• Let $C$ be a vertex cover of size $k = d + 2m$ in $G$.

• Since it is a vertex cover, it must include at least two out of three nodes in each "clause gadget" at the bottom.
Example

- Since it is a vertex cover, it must include at least two out of three nodes in each "clause gadget" at the bottom.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
Example

- Since it is a vertex cover, it must include at least two out of three nodes in each "clause gadget" at the bottom.

Running example: \( \phi = (x_1 \lor x_1 \lor x_2) \land (\lnot x_1 \lor \lnot x_2 \lor \lnot x_2) \land (\lnot x_1 \lor x_2 \lor x_2) \)
Other direction

- G has a vertex cover of size at most $k$. \(\Rightarrow\) \(\phi\) is satisfiable.

- Let \(C\) be a vertex cover of size \(k = d + 2m\) in \(G\).

- Since it is a vertex cover, it must include at least two out of three nodes in each “clause gadget” at the bottom.
Other direction

- $G$ has a vertex cover of size at most $k$. $\Rightarrow \phi$ is satisfiable.

- Let $C$ be a vertex cover of size $k = d + 2m$ in $G$.

- Since it is a vertex cover, it must include at least two out of three nodes in each “clause gadget” at the bottom.

- This means that at least $2m$ nodes of $C$ are at the bottom.
Other direction

• G has a vertex cover of size at most $k$. $\Rightarrow \phi$ is satisfiable.

• Let C be a vertex cover of size $k = d + 2m$ in $G$.

• Since it is a vertex cover, it must include at least two out of three nodes in each "clause gadget" at the bottom.

• This means that at least $2m$ nodes of $C$ are at the bottom.

• This means that at most $d$ nodes of $C$ are at the top.
Other direction

- This means that at most \( d \) nodes of \( C \) are at the top.

- To satisfy the edges at the top, in each "variable gadget", at least one node must be included in \( C \).
Example

- To satisfy the edges at the top, in each "variable gadget", at least one node must be included in C.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
Example

- To satisfy the edges at the top, in each “variable gadget”, at least one node must be included in $C$.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
Other direction

• This means that at most $d$ nodes of $C$ are at the top.

• To satisfy the edges between the top and the bottom, in each “variable gadget”, at least one node must be included in $C$. 
Other direction

- This means that at most $d$ nodes of $C$ are at the top.

- To satisfy the edges between the top and the bottom, in each “variable gadget”, at least one node must be included in $C$.

- From the two statements above, in each “variable gadget”, exactly one node must be included in $C$. 
Satisfying the formula
Satisfying the formula

- Consider the truth assignment corresponding to the nodes of the vertex cover $C$ on the top (in the variable gadgets).
Satisfying the formula

- Consider the truth assignment corresponding to the nodes of the vertex cover $C$ on the top (in the variable gadgets).

- Note that we either choose $x_i$ or $\overline{x_i}$ to be 1, but not both.
Satisfying the formula

• Consider the truth assignment corresponding to the nodes of the vertex cover $C$ on the top (in the variable gadgets).

• Note that we either choose $x_i$ or $\overline{x_i}$ to be 1, but not both.

  • From the statement “in each “variable gadget”, exactly one node must be included in $C$”.

Satisfying the formula

• Consider the truth assignment corresponding to the nodes of the vertex cover C on the top (in the variable gadgets).

• Note that we either choose $x_i$ or $\overline{x}_i$ to be 1, but not both.
  
  • From the statement “in each “variable gadget”, exactly one node must be included in C”.

• Since all “cross” edges are covered, there must be one endpoint on the top (in the “variable gadget”) that is in C.
Satisfying the formula

- Consider the truth assignment corresponding to the nodes of the vertex cover C on the top (in the variable gadgets).

- Note that we either choose $x_i$ or $\overline{x}_i$ to be 1, but not both.
  - From the statement “in each “variable gadget”, exactly one node must be included in C”.

- Since all “cross” edges are covered, there must be one endpoint on the top (in the “variable gadget”) that is in C.
  - This means that there is one variable of the clause that is set to 1.
Satisfying the formula

- Consider the truth assignment corresponding to the nodes of the vertex cover C on the top (in the variable gadgets).

- Note that we either choose $x_i$ or $\overline{x_i}$ to be 1, but not both.
  - From the statement “in each “variable gadget”, exactly one node must be included in C”.

- Since all “cross” edges are covered, there must be one endpoint on the top (in the “variable gadget”) that is in C.
  - This means that there is one variable of the clause that is set to 1.
  - Thus the clause is satisfied.
Example

- To satisfy the edges at the top, in each “variable gadget”, at least one node must be included in C.

Running example: $\phi = (x_1 \lor x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_2)$
Vertex Cover

• **Definition:** A vertex cover $C$ of a graph $G=(V, E)$ is a subset of the nodes such that every edge $e$ in the graph has at least one endpoint in $C$.

• **Definition:** A minimum vertex cover is a vertex cover of the smallest possible size.

• **Vertex Cover**
  
  **Input:** A graph $G=(V, E)$
  
  **Output:** A minimum vertex cover.
Vertex Cover
decision version

• **Definition:** A vertex cover $C$ of a graph $G=(V, E)$ is a subset of the nodes such that every edge $e$ in the graph has at least one endpoint in $C$.

• **Definition:** A minimum vertex cover is a vertex cover of the smallest possible size.

• **Vertex Cover**
  **Input:** A graph $G=(V, E)$ and a number $k$
  **Output:** Is there a vertex cover of size $\leq k$?
From optimisation to decision

• We are given an optimisation problem $P$ (assume minimisation).

  • E.g., find the minimum vertex cover.

• We introduce a threshold $k$.

• The decision version $P_d$ becomes: Given an instance of $P$ and the threshold $k$ as input, is there a solution to $P$ of value at most $k$?

  • E.g., is there a vertex cover of size at most $k$?
Optimisation vs decision
Optimisation vs decision

- If we can solve $P$ in polynomial time, we can solve $P_d$ in polynomial time. (why?)
Optimisation vs decision

• If we can solve $P$ in polynomial time, we can solve $P_d$ in polynomial time. (why?)

• This implies that if the decision version is NP-hard, so is the optimisation version.
• If we can solve $P$ in polynomial time, we can solve $P_d$ in polynomial time. (why?)

• This implies that if the decision version is NP-hard, so is the optimisation version.

• Note: It is generally not correct to say that an optimisation problem is NP-complete!
Optimisation vs decision

• If we can solve $P$ in polynomial time, we can solve $P_d$ in polynomial time. \(\text{why?}\)

• This implies that if the decision version is \text{NP-hard}, so is the optimisation version.

• Note: It is generally not correct to say that an optimisation problem is \text{NP-complete}!

• Often the opposite is also true.
Optimisation vs decision

• If we can solve $P$ in polynomial time, we can solve $P_d$ in polynomial time. (why?)

• This implies that if the decision version is NP-hard, so is the optimisation version.

• Note: It is generally not correct to say that an optimisation problem is NP-complete!

• Often the opposite is also true.

• If we can solve $P_d$ in polynomial time, we can solve $P$ in polynomial time.
Optimisation vs decision

- **Vertex Cover (Optimisation)**
  - **Input:** A graph $G=(V, E)$
  - **Output:** A minimum vertex cover.

- **Vertex Cover (Decision)**
  - **Input:** A graph $G=(V, E)$ and a number $k$
  - **Output:** Is there a vertex cover of size $\leq k$?
Optimisation vs decision

- **Vertex Cover Size (Optimisation)**
  
  *Input*: A graph $G = (V, E)$
  
  *Output*: The size of a minimum vertex cover.

- **Vertex Cover (Decision)**
  
  *Input*: A graph $G = (V, E)$ and a number $k$
  
  *Output*: Is there a vertex cover of size $\leq k$?
Vertex Cover Size

VC (decision)
Vertex Cover Size

$k = 1$?

VC (decision)
Vertex Cover Size

$k = 1$?

no

VC (decision)
Vertex Cover Size

$k = 1$?

no

$k = 2$?

VC (decision)
Vertex Cover Size

k = 1 ?
no

k = 2 ?
no

VC (decision)
Vertex Cover Size

- $k = 1$?
  - no
  - $k = 2$?
  - no
  - ...

VC (decision)
Vertex Cover Size

$k = 1$?

no

$k = 2$?

no

...  

$k = n$?

VC (decision)
Vertex Cover Size

\[ k = 1 ? \]
\[ \text{no} \]
\[ k = 2 ? \]
\[ \text{no} \]
\[ \ldots \]
\[ k = n ? \]
\[ \text{yes} \]
Vertex Cover Size

\[ k = 1 \text{ ?} \rightarrow \text{no} \]
\[ k = 2 \text{ ?} \rightarrow \text{no} \]
\[ \ldots \]
\[ k = n \text{ ?} \rightarrow \text{yes} \]

VC (decision)
Vertex Cover Size

k = 1 ?
no

k = 2 ?
no

k = l-1 ?
no

k = l ?
yes

k = n ?
yes

VC (decision)
Vertex Cover Size

VC (decision)
Vertex Cover Size

$k = 1$?
Vertex Cover Size

k = 1 ?

no

VC (decision)
Vertex Cover Size

k = 1 ?
no

k = n ?

VC (decision)
Vertex Cover Size

$k = 1$?

- no

$k = n$?

- yes

VC (decision)
Vertex Cover Size

- k = 1?
  - no

- k = n/2?

- k = n?
  - yes
Vertex Cover Size

\[ k = 1 ? \]
\[ \text{no} \]

\[ k = n/2 ? \]
\[ \text{no} \]

\[ k = n ? \]
\[ \text{yes} \]
Vertex Cover Size

- $k = 1$ ?
  - no

- $k = n/2$ ?
  - no

- $k = n$ ?
  - yes
Optimisation vs decision

• **Vertex Cover Size (Optimisation)**
  - **Input:** A graph $G=(V, E)$
  - **Output:** The size of a minimum vertex cover.

• **Vertex Cover (Decision)**
  - **Input:** A graph $G=(V, E)$ and a number $k$
  - **Output:** Is there a vertex cover of size $\leq k$?
Optimisation vs decision

- **Vertex Cover (Optimisation)**
  
  **Input:** A graph $G=(V, E)$
  
  **Output:** A minimum vertex cover.

- **Vertex Cover (Decision)**
  
  **Input:** A graph $G=(V, E)$ and a number $k$
  
  **Output:** Is there a vertex cover of size $\leq k$?
Vertex Cover
Vertex Cover

• First, find the value $k^*$ of the minimum vertex cover using the algorithm for $\text{VC}_d$. 
Vertex Cover

• First, find the value $k^*$ of the minimum vertex cover using the algorithm for $VC_d$.

• Pick a vertex $v$ in the graph.
Vertex Cover

• First, find the value $k^*$ of the minimum vertex cover using the algorithm for $\text{VC}_d$.

• Pick a vertex $v$ in the graph.
  
  • Remove it (and the incident edges) to get graph $G - \{v\}$. 
Vertex Cover

• First, find the value $k^*$ of the minimum vertex cover using the algorithm for $VC_d$.

• Pick a vertex $v$ in the graph.
  
  • Remove it (and the incident edges) to get graph $G - \{v\}$.

  • Property: If $v$ was in any minimum vertex cover, $G - \{v\}$ has a minimum vertex cover of size $k^*-1$. 
Vertex Cover

• First, find the value $k^*$ of the minimum vertex cover using the algorithm for $VC_d$.

• Pick a vertex $v$ in the graph.

  • Remove it (and the incident edges) to get graph $G - \{v\}$.

  • Property: If $v$ was in any minimum vertex cover, $G - \{v\}$ has a minimum vertex cover of size $k^*-1$.

  • Check if the graph $G - \{v\}$ has a vertex cover of size at most $k^*-1$. 
Vertex Cover

• First, find the value $k^*$ of the minimum vertex cover using the algorithm for $VC_d$.

• Pick a vertex $v$ in the graph.
  
  • Remove it (and the incident edges) to get graph $G - \{v\}$.
  
  • **Property:** If $v$ was in any minimum vertex cover, $G - \{v\}$ has a minimum vertex cover of size $k^* - 1$.
  
  • Check if the graph $G - \{v\}$ has a vertex cover of size at most $k^* - 1$.
    
    • **Yes:** Include $v$ in the vertex cover.
Vertex Cover

- First, find the value $k^*$ of the minimum vertex cover using the algorithm for $VC_d$.

- Pick a vertex $v$ in the graph.
  - Remove it (and the incident edges) to get graph $G - \{v\}$.
  - **Property:** If $v$ was in any minimum vertex cover, $G - \{v\}$ has a minimum vertex cover of size $k^*-1$.
  - Check if the graph $G - \{v\}$ has a vertex cover of size at most $k^*-1$.
    - **Yes:** Include $v$ in the vertex cover.
    - **No:** Do not include $v$ in the vertex cover.
Vertex Cover

• First, find the value \( k^* \) of the minimum vertex cover using the algorithm for \( VC_d \).

• Pick a vertex \( v \) in the graph.
  
  • Remove it (and the incident edges) to get graph \( G - \{v\} \).
  
  • Property: If \( v \) was in any minimum vertex cover, \( G - \{v\} \) has a minimum vertex cover of size \( k^*-1 \).
  
  • Check if the graph \( G - \{v\} \) has a vertex cover of size at most \( k^*-1 \).
    
    • Yes: Include \( v \) in the vertex cover.
    
    • No: Do not include \( v \) in the vertex cover.
  
  • Then move to the next vertex.
The subset sum problem

- We are given a set of \( n \) items \( \{1, 2, \ldots, n\} \).
- Each item \( i \) has a non-negative integer weight \( w_i \).
- We are given an integer bound \( W \).
- Goal: Select a subset \( S \) of the items such that \( \sum_{i \in S} w_i \leq W \) and \( \sum_{i \in S} w_i \) is maximised.
Equivalent formulation
decision version

• We are given a set \( T \) of \( n \) items \( \{1, 2, \ldots, n\} \).

• Each item \( i \) has a non-negative integer weight \( w_i \).

• We are given an integer bound \( W \).

• Goal: Decide if there exists a subset \( S \) of the items such that

\[
\sum_{i \in S} w_i = W
\]
Optimisation vs decision

- If we can solve $P$ in polynomial time, we can solve $P_d$ in polynomial time. (why?)
  - This implies that if the decision version is NP-hard, so is the optimisation version.
- Often the opposite is also true.
  - If we can solve $P_d$ in polynomial time, we can solve $P$ in polynomial time.
Optimisation vs decision

• If we can solve $P$ in polynomial time, we can solve $P_d$ in polynomial time. (*why?*)

  • This implies that if the decision version is NP-hard, so is the optimisation version.

• Often the opposite is also true.

  • If we can solve $P_d$ in polynomial time, we can solve $P$ in polynomial time.

*We did this for VC. Can we also do it for SS?*