Advanced Algorithmic Techniques (COMP523)

Approximation Algorithms 2

Recap and plan

- **Previously lecture:**
  - Approximation algorithms: approach and challenges.
  - Greedy method
    - Application: Load Balancing on identical machines.
  - Approximation Ratio.
- **This lecture:**
  - The Pricing Method.
    - Application: Vertex Cover.

Methods for approximation algorithms

- Greedy algorithms.
- Pricing method (also known as the Primal-Dual method).
- Linear Programming and Rounding.
- Dynamic Programming on rounded inputs.

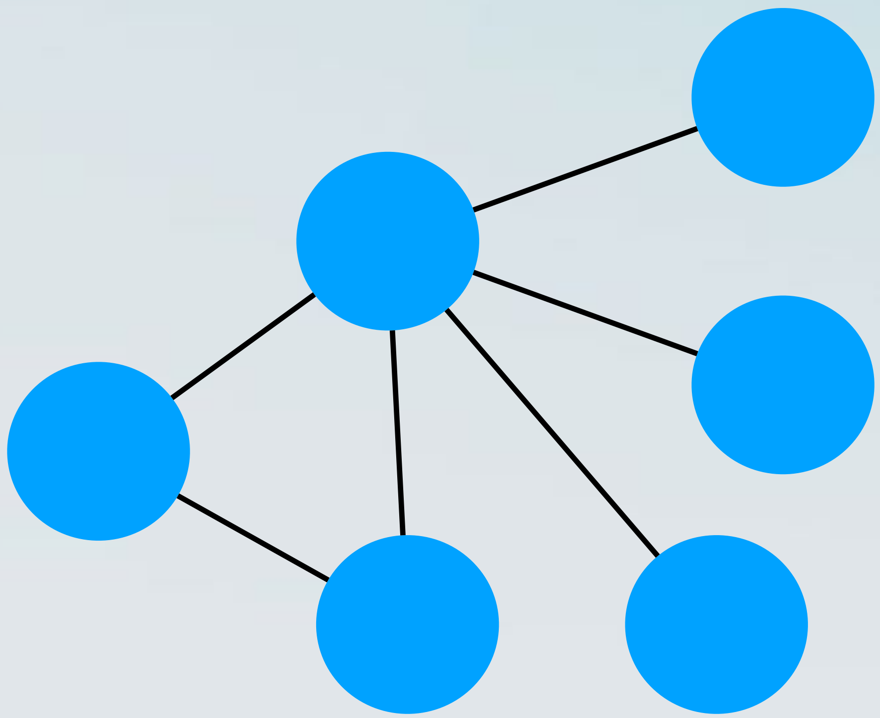
Vertex Cover

- 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.

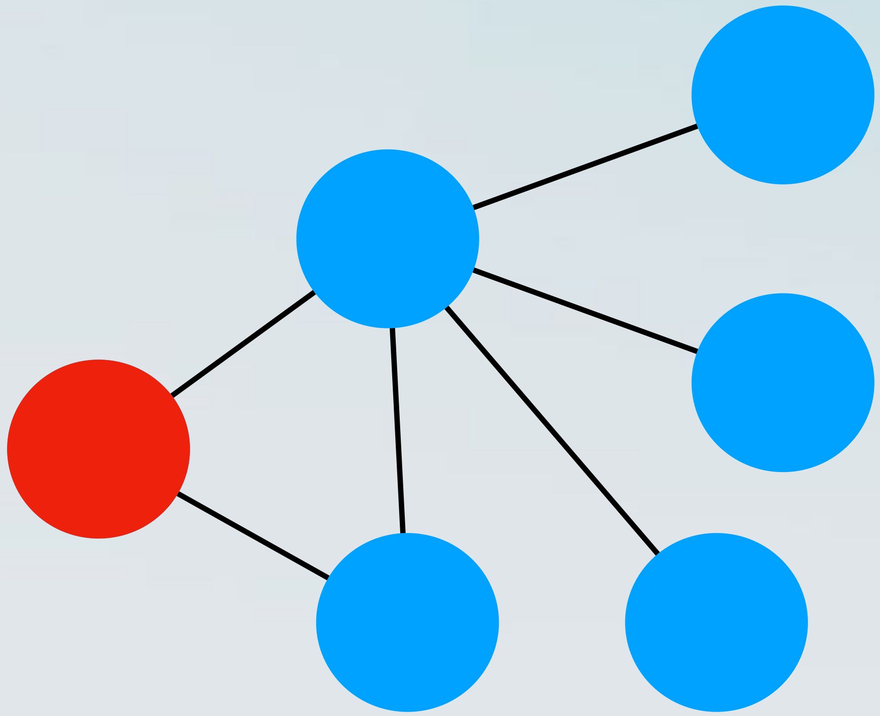
- Definition: A minimum vertex cover is a vertex cover of the smallest possible size.

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

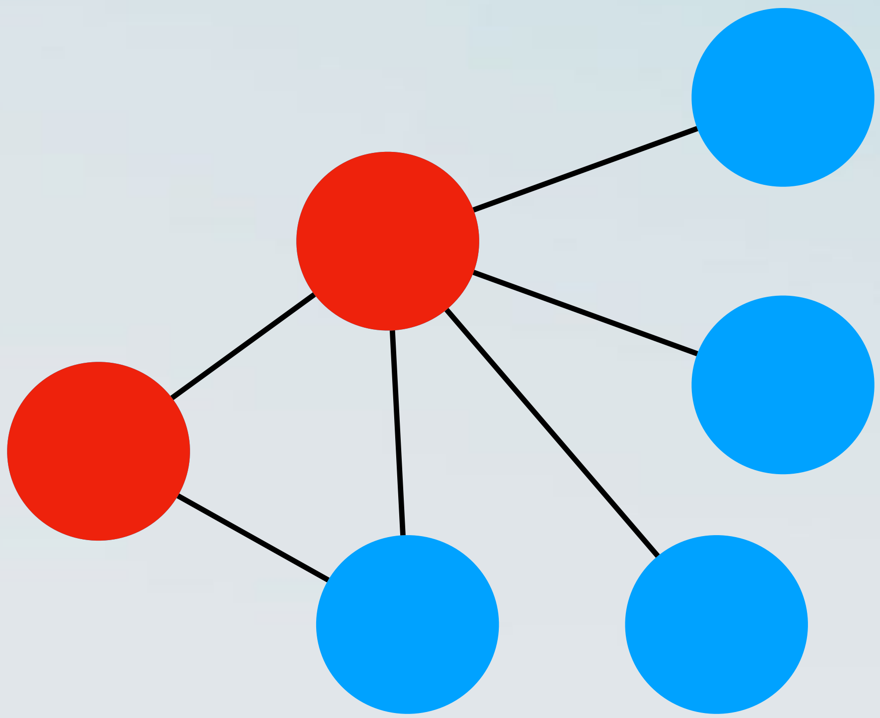
Example



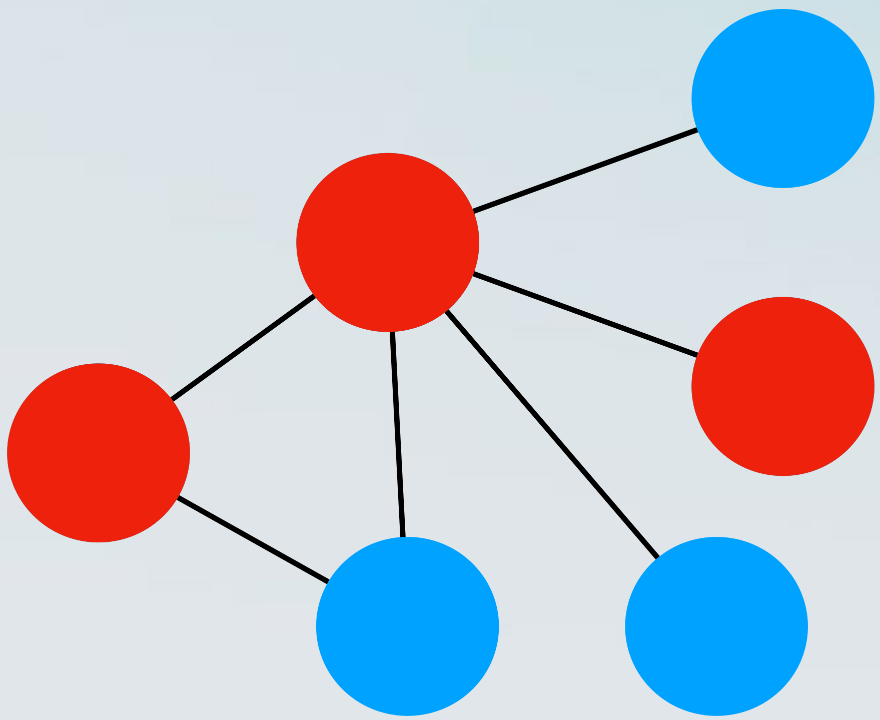
Example



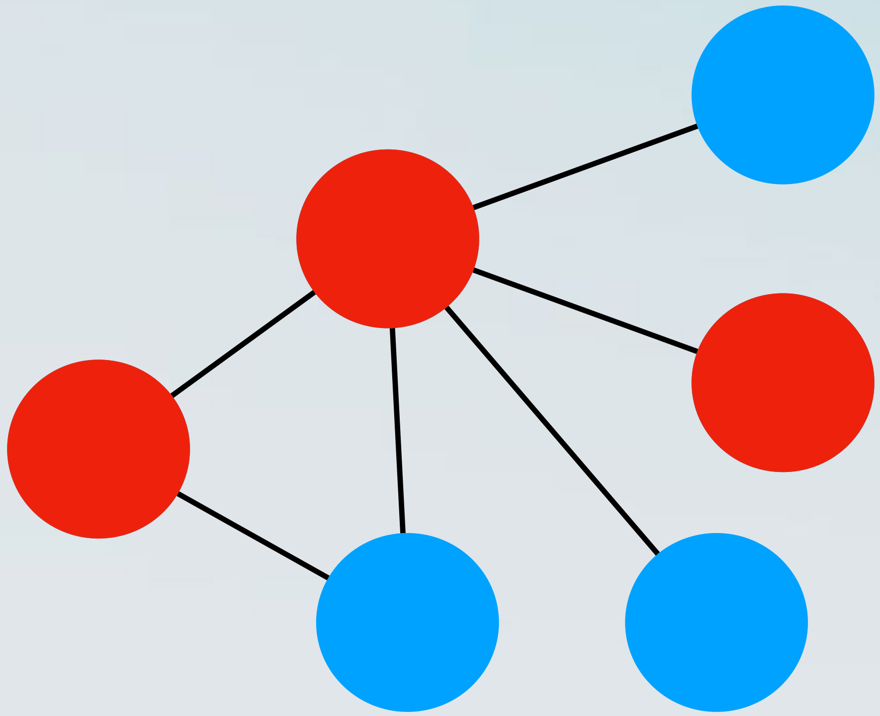
Example



Example

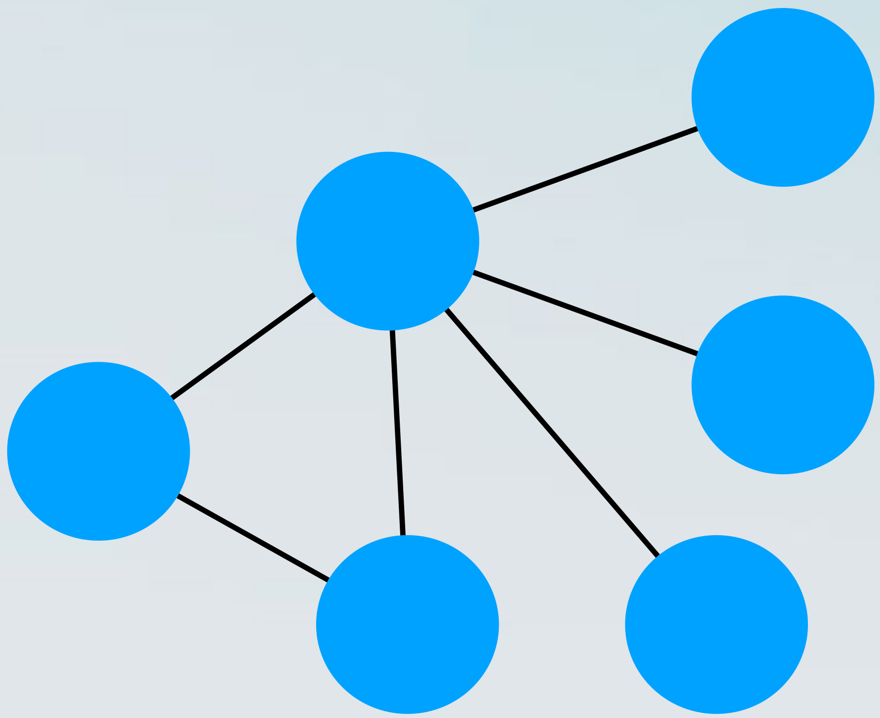


Example

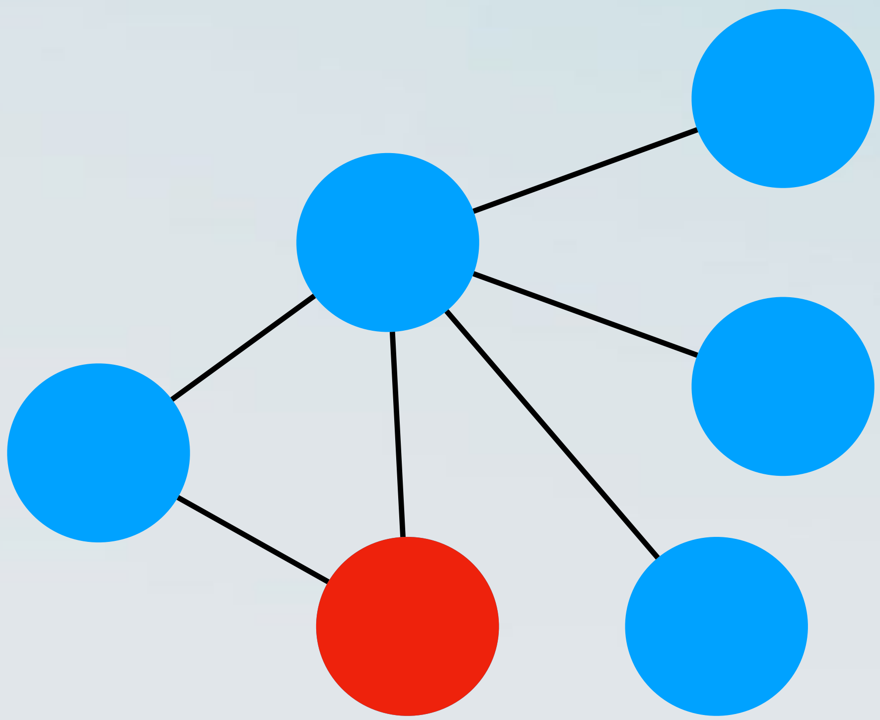


A vertex cover

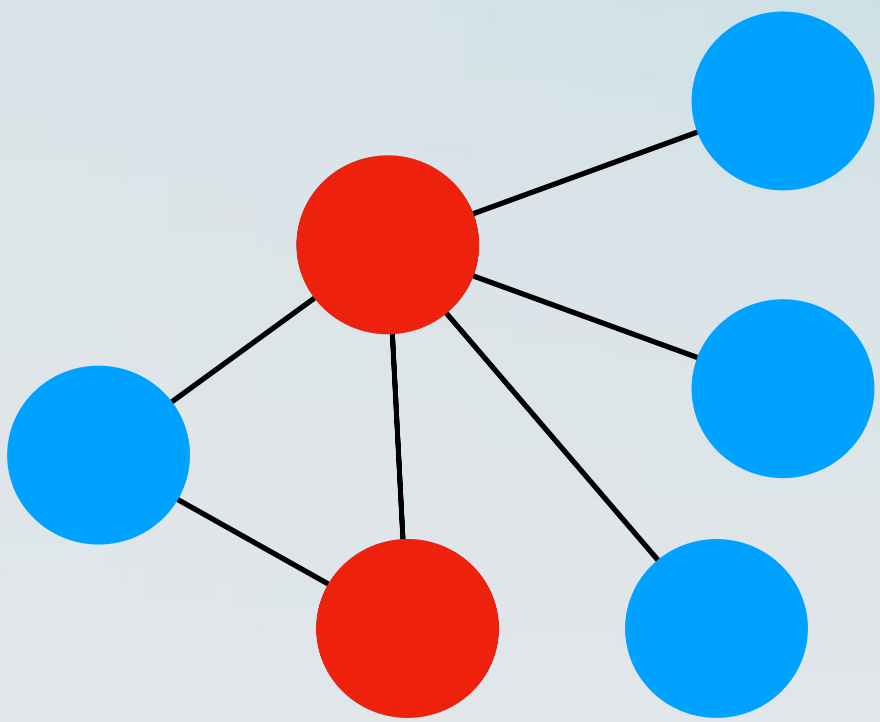
Example



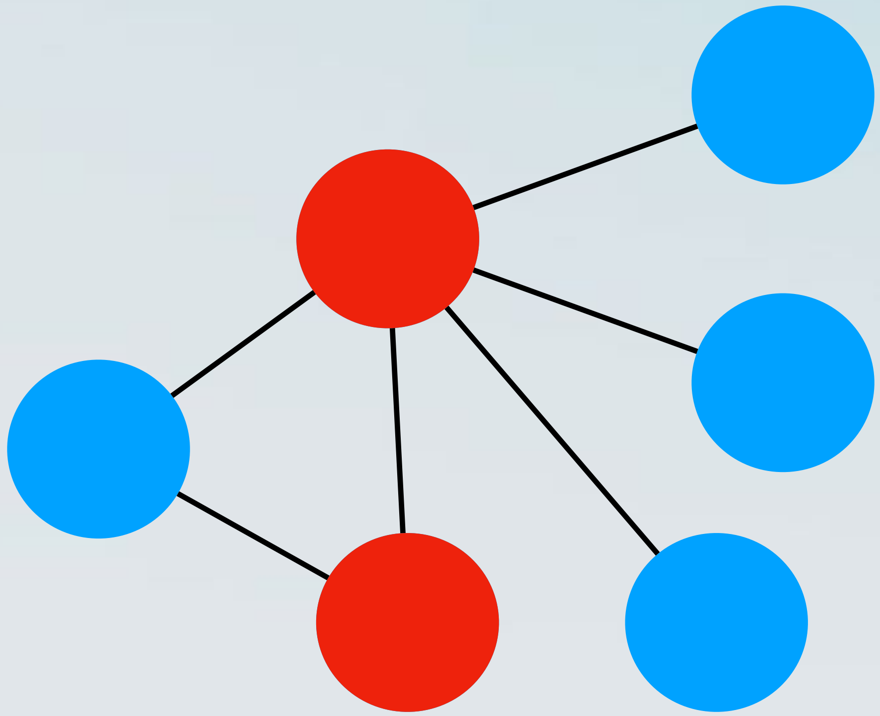
Example



Example



Example



A minimum vertex cover

Vertex Cover

- We proved that Vertex Cover is NP-complete.

Vertex Cover

- We proved that Vertex Cover is NP-complete.
  - To be more precise:

Vertex Cover

- We proved that Vertex Cover is NP-complete.
  - To be more precise:
  - We proved that the *decision version* is NP-complete.

Vertex Cover

- We proved that Vertex Cover is NP-complete.
  - To be more precise:
  - We proved that the *decision version* is NP-complete.
  - This implies that the *optimisation version* is NP-hard.

Vertex Cover

- We proved that Vertex Cover is NP-complete.
  - To be more precise:
  - We proved that the *decision version* is NP-complete.
  - This implies that the *optimisation version* is NP-hard.
- This means that we can not hope to solve it optimally in polynomial time.

Vertex Cover

- We proved that Vertex Cover is NP-complete.
  - To be more precise:
  - We proved that the *decision version* is NP-complete.
  - This implies that the *optimisation version* is NP-hard.
- This means that we can not hope to solve it optimally in polynomial time.
- Can we solve it approximately?

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.

- Each vertex i has a weight w_i.

- Definition: A minimum vertex cover is a vertex cover of the smallest possible total weight.

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

Vertex Cover

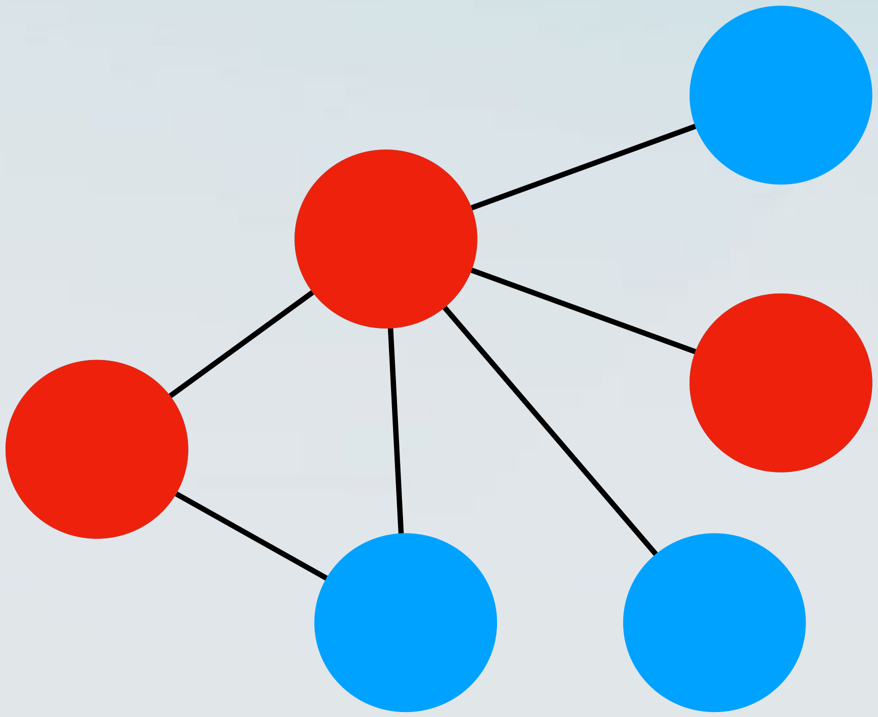
- Definition: A vertex cover C of a graph G=(V, E) is a subset of the nodes such that every edge e in the graph has at least one endpoint in C.
- Each vertex i has a weight w_i.
- Definition: A minimum vertex cover is a vertex cover of the smallest possible total weight.
- Vertex Cover
  Input: A graph G=(V, E)
  Output: A minimum (weight) vertex cover.
- If we can solve the weighted version of vertex cover, we can solve the unweighted version (why?)

Vertex Cover

- We will design a polynomial time approximation algorithm for the weighted vertex cover problem.

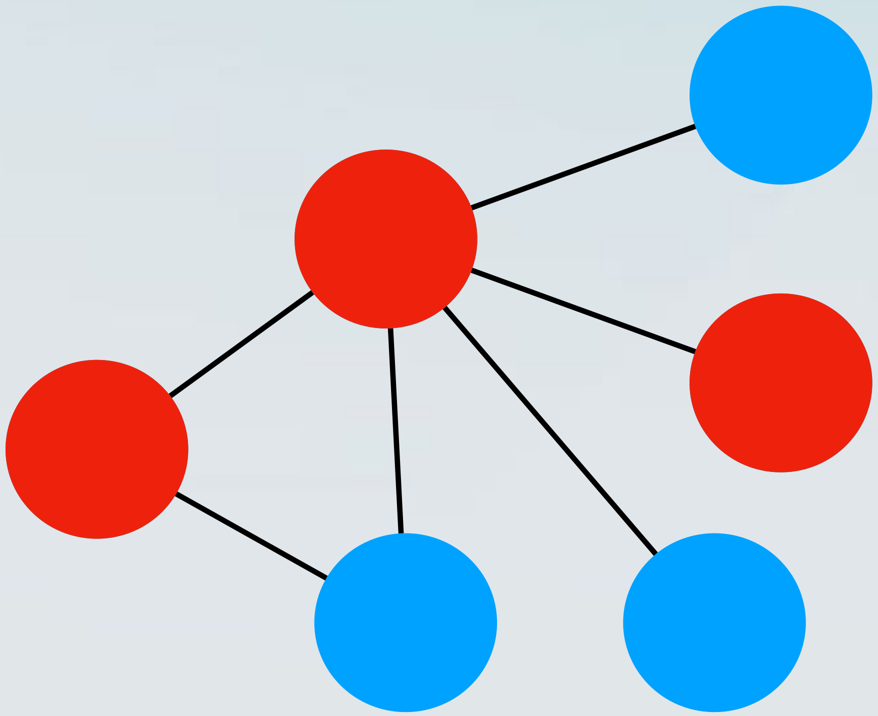
The Pricing Method

- Consider a vertex cover S.



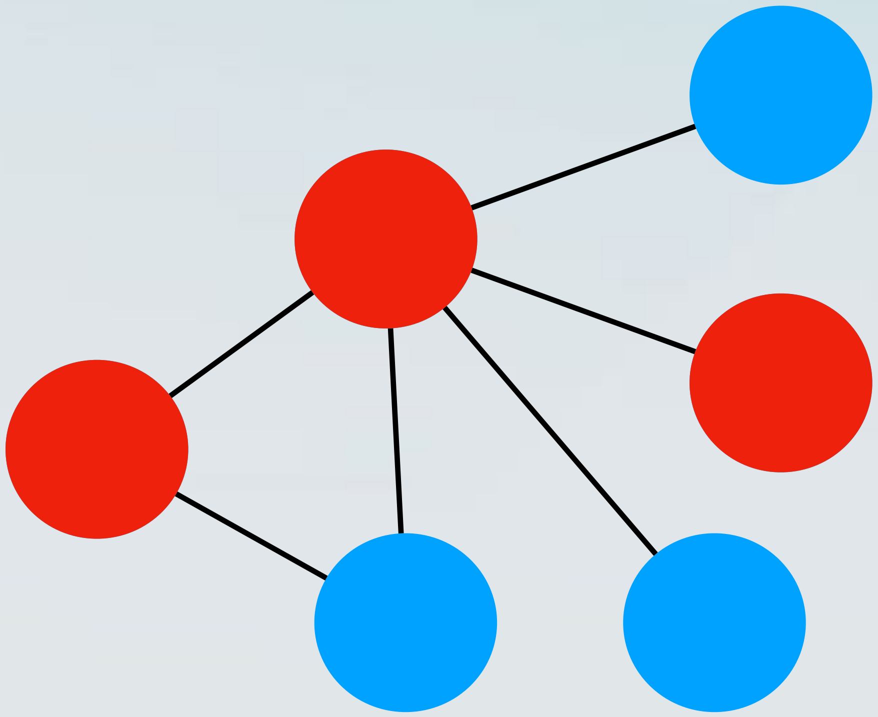
The Pricing Method

- Consider a vertex cover S.
- Every vertex i has a *cost*, which is equal to w_i.



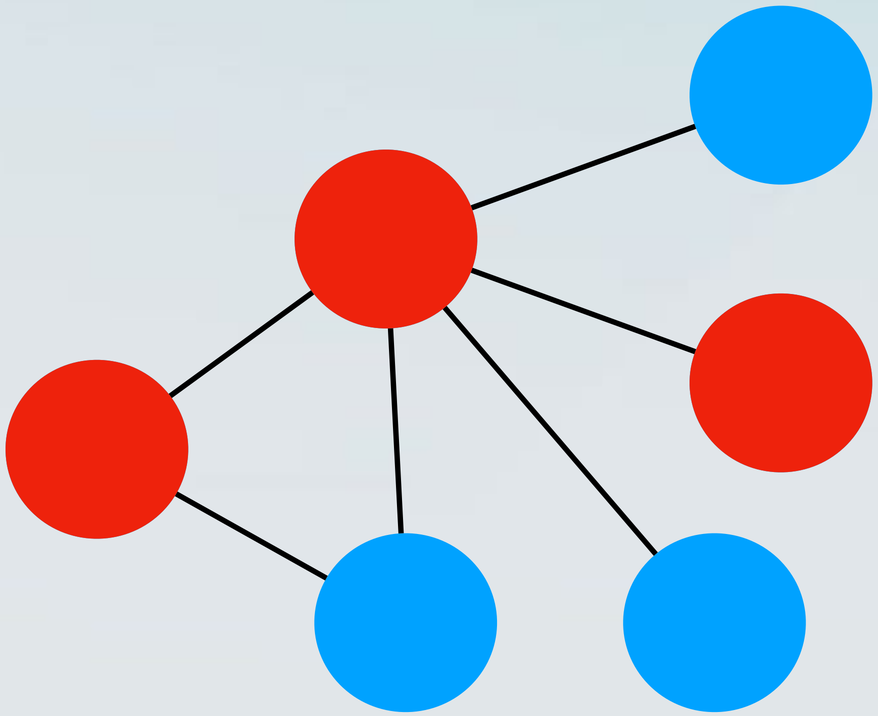
The Pricing Method

- Consider a vertex cover S.
- Every vertex i has a *cost*, which is equal to w_i.
- If i is included in S, the edges that "*use it*" have to pay for (some of) the cost.



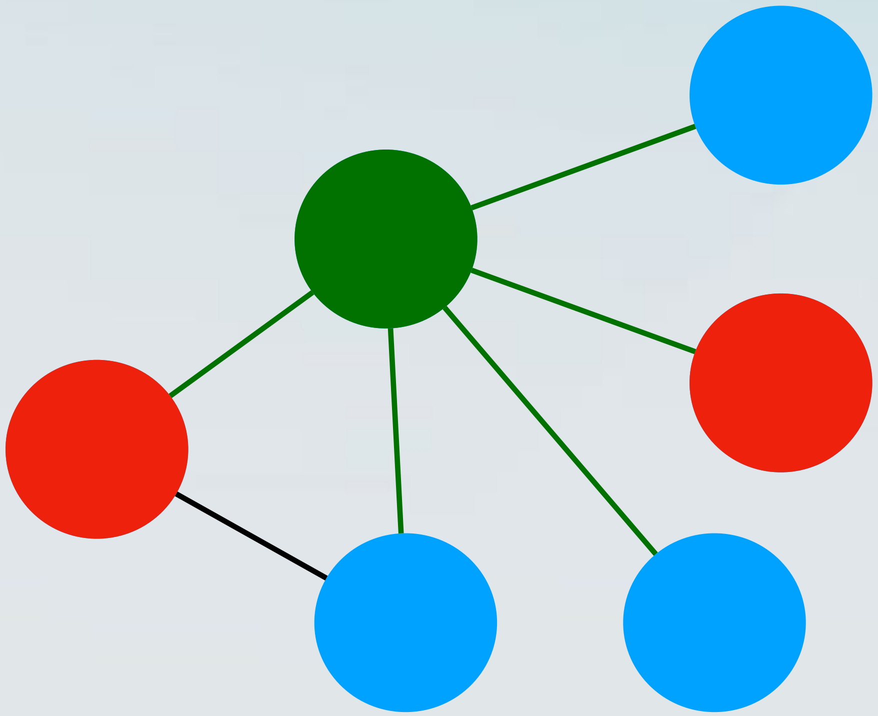
The Pricing Method

- Consider a vertex cover S.
- Every vertex i has a *cost*, which is equal to w_i.
- If i is included in S, the edges that "*use it*" have to pay for (some of) the cost.
- Each edge e in the graph pays a *price* p_e.



The Pricing Method

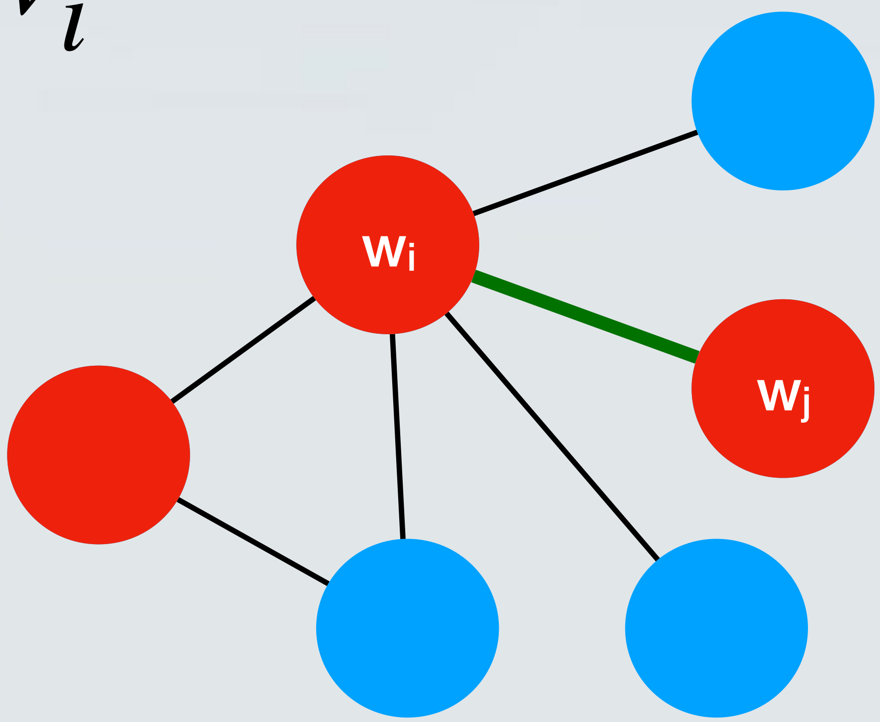
- Consider a vertex cover S.
- Every vertex i has a *cost*, which is equal to w_i.
- If i is included in S, the edges that "*use it*" have to pay for (some of) the cost.
- Each edge e in the graph pays a *price* p_e.



Fair Pricing

- Given a vertex i, we never ask the edges that "*use it*" to pay more than the cost of the vertex.

$$\sum_{e=(i,j)} p_e \leq w_i$$



Fair Pricing Lemma

- Lemma: Let S be any vertex cover and let p_e be any non-negative fair prices. Then, it holds that:

$$\sum_{e \in E} p_e \leq w(S)$$

Proof of the lemma

Proof of the lemma

- By fairness, we have that: $\sum_{e=(i,j)} p_e \leq w_i$

Proof of the lemma

- By fairness, we have that: $\sum_{e=(i,j)} p_e \leq w_i$
- Adding up over all nodes, we have:

$$\sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i = w(S)$$

Proof of the lemma

- By fairness, we have that: $\sum_{e=(i,j)} p_e \le w_i$
- Adding up over all nodes, we have:

$$\sum_{i \in S} \sum_{e=(i,j)} p_e \le \sum_{i \in S} w_i = w(S)$$

- Let's look at this expression.

Proof of the lemma

- By fairness, we have that: $\sum_{e=(i,j)} p_e \leq w_i$
- Adding up over all nodes, we have:

$$\sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i = w(S)$$

- Let's look at this expression.
- Since S is a vertex cover, each edge contributes *at least* on term p_e to the expression.

Proof of the lemma

- By fairness, we have that: $\sum_{e=(i,j)} p_e \leq w_i$
- Adding up over all nodes, we have:

$$\sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i = w(S)$$

- Let's look at this expression.
- Since S is a vertex cover, each edge contributes *at least* on term p_e to the expression.
- From this, we get that: $\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e$

The algorithm

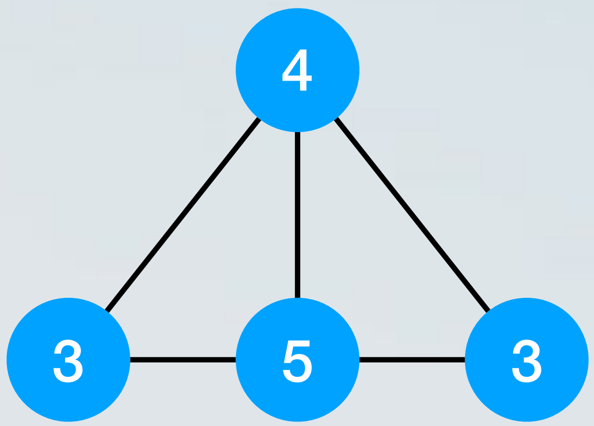
- Terminology: We will say that node i is "*tight*" if

$$\sum_{e=(i,j)} p_e = w_i$$

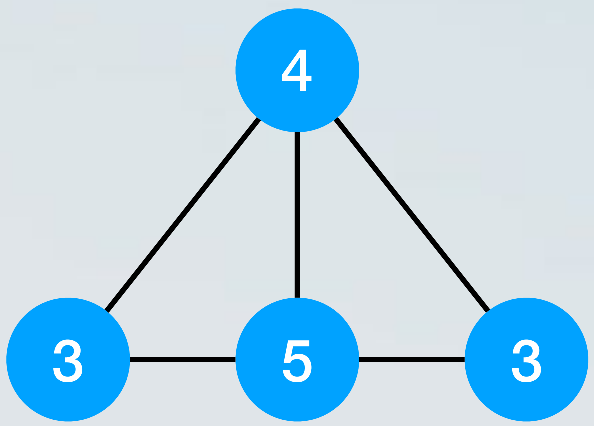
Vertex-Cover-Approx(G,w)

Set p_e = *0* for all e in E.
While there is an edge e=(i, j) such that neither i nor j is tight
    Select such an edge e
    Increase p_e *without violating fairness*
EndWhile
Let S be the *set of all tight nodes*
Return S.

Example

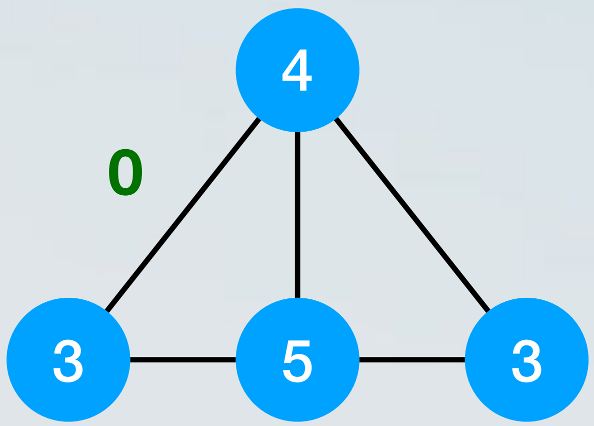


Example



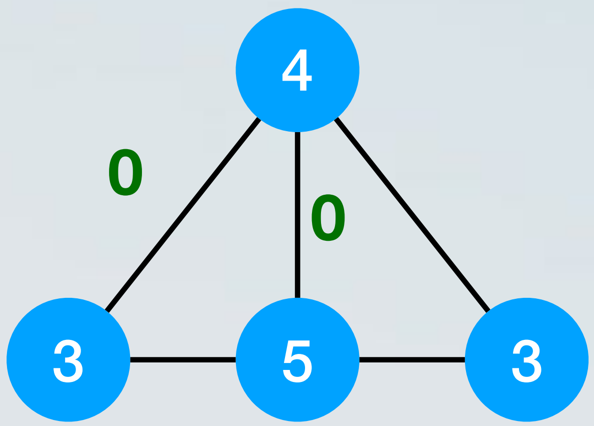
Set $p_e = 0$ for all e in E.

Example



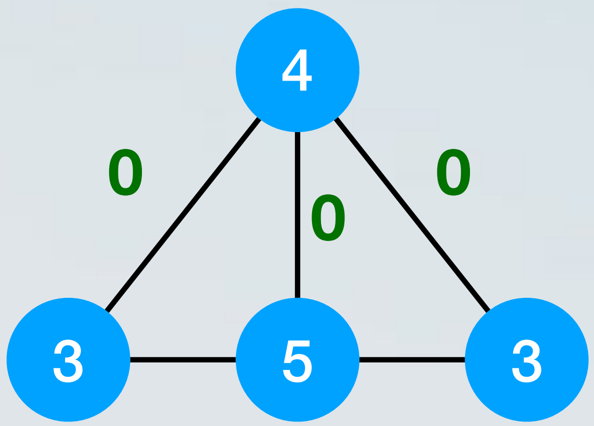
Set $p_e = 0$ for all e in E.

Example



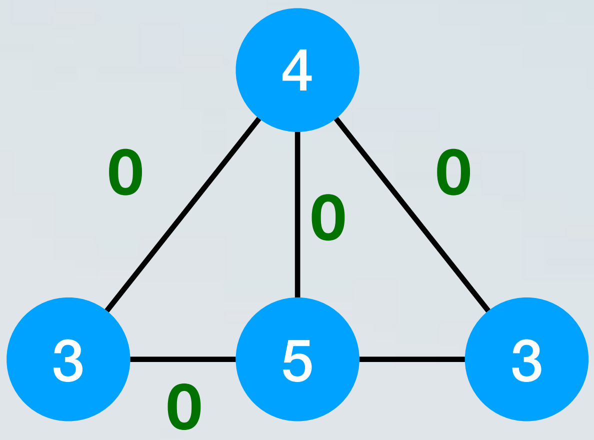
Set $p_e = 0$ for all e in E.

Example



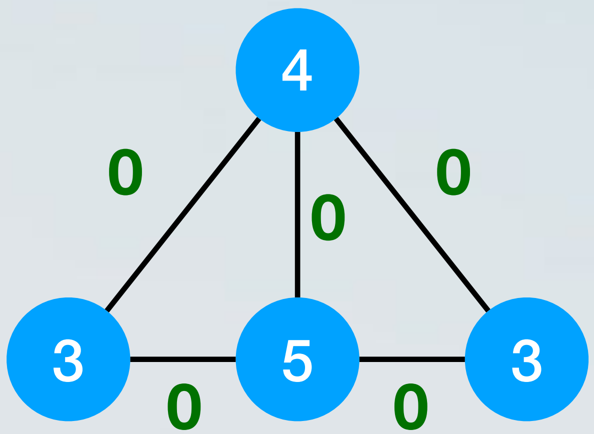
Set $p_e = 0$ for all e in E.

Example



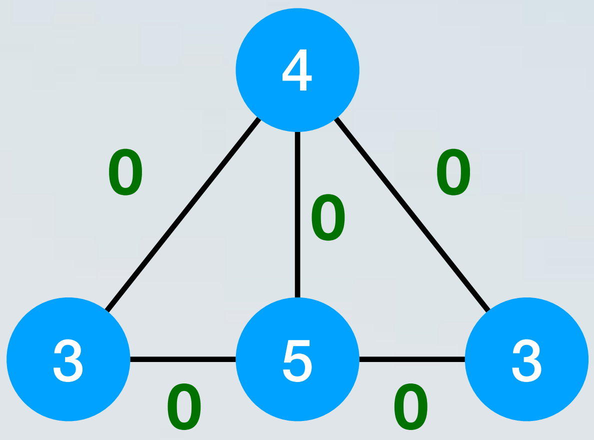
Set $p_e = 0$ for all e in E.

Example



Set $p_e = 0$ for all e in E.

Example

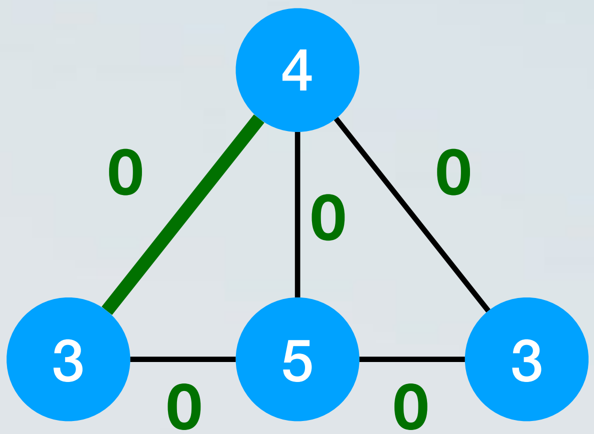


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight

Select such an edge e

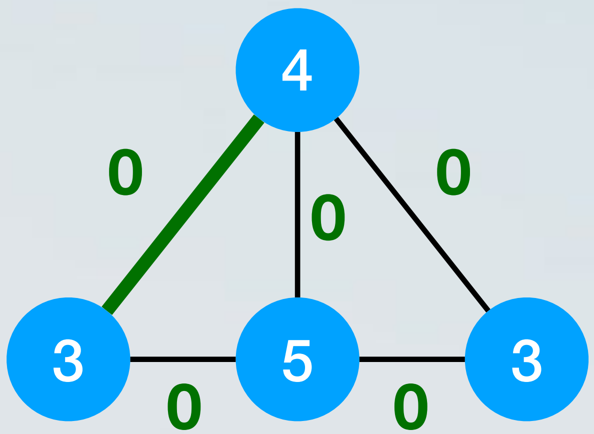
Example



Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Example

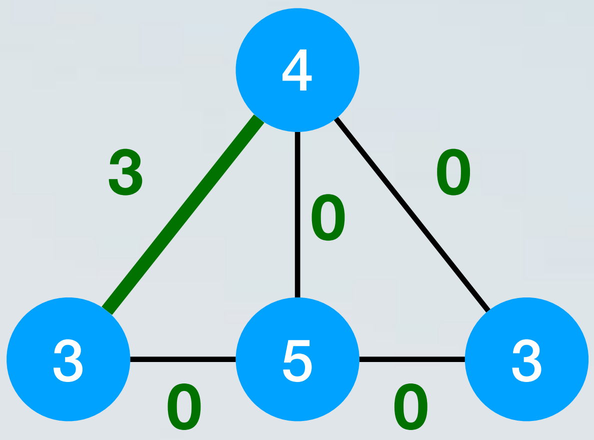


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example

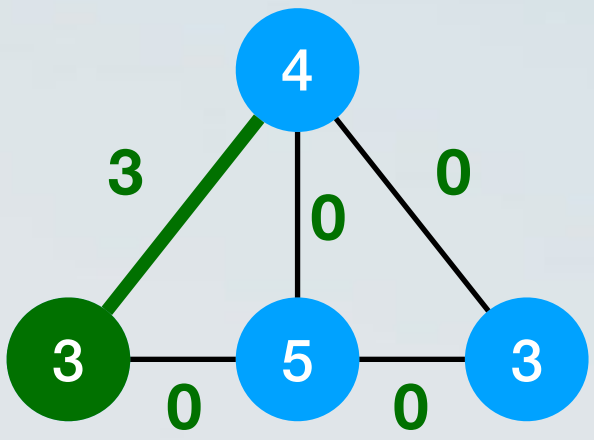


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example

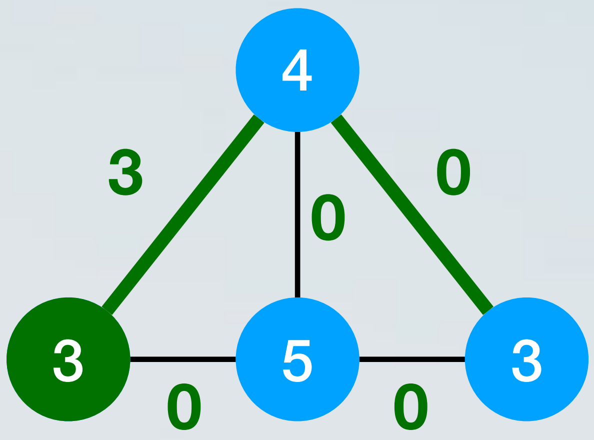


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example

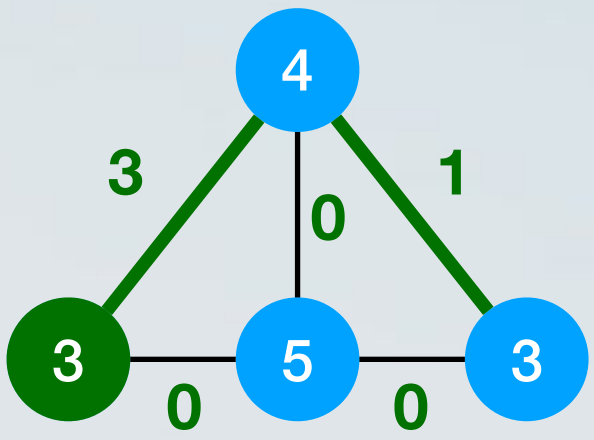


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example

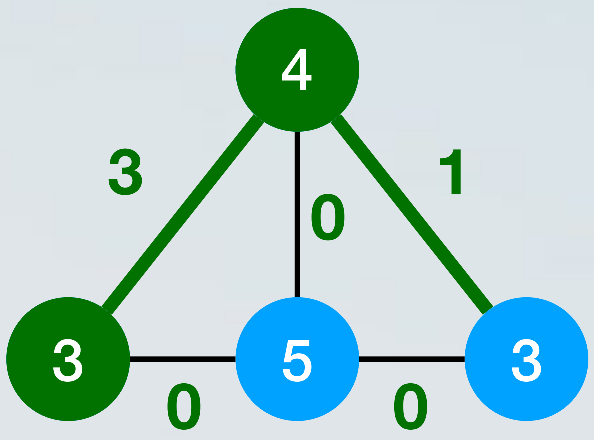


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example

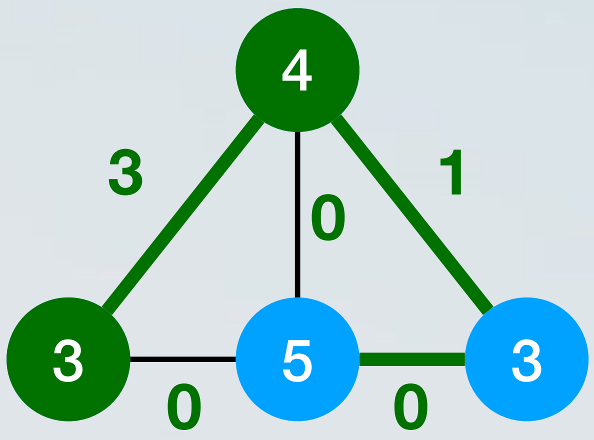


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example

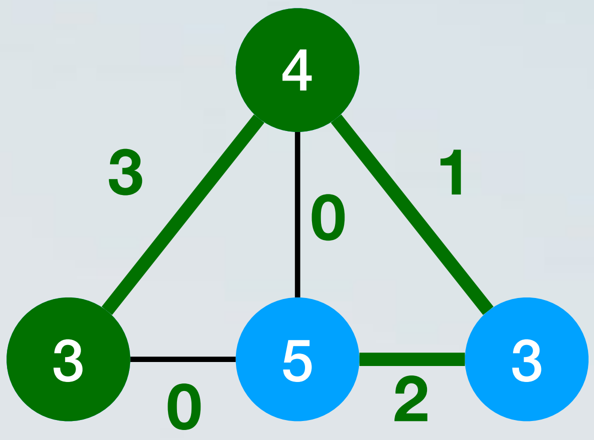


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example

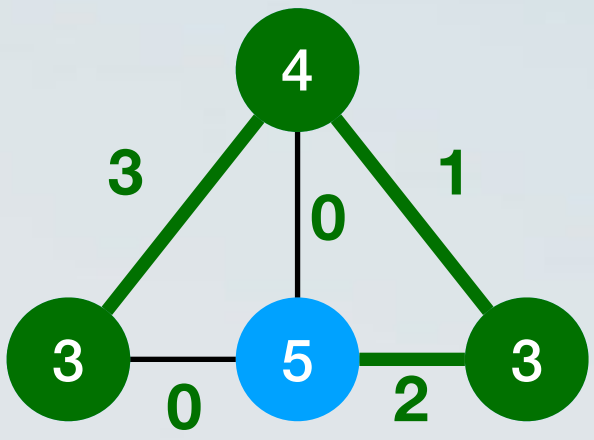


Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example

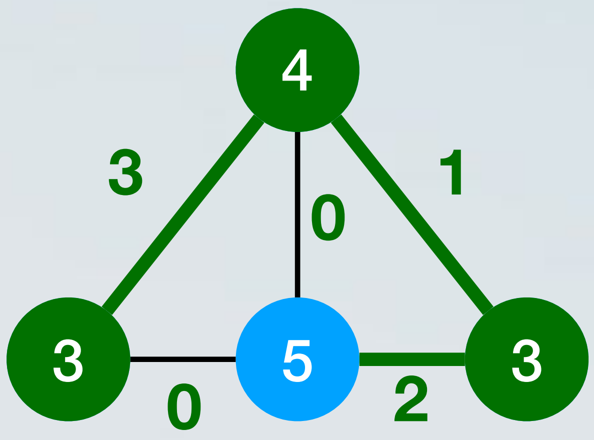


Set $p_e = 0$ for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Example



Set p_e = *0* for all e in E.

While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e

Increase p_e *without violating fairness*

Let S be the *set of all tight nodes*
Return S

Intuition

Intuition

- Consider any node i in the vertex cover S.

Intuition

- Consider any node i in the vertex cover S.
- Since i is *tight*, its cost is exactly covered by the edges that are incident to it.

Intuition

- Consider any node i in the vertex cover S.
- Since i is *tight*, its cost is exactly covered by the edges that are incident to it.
- However, an edge (i, j) might be incident to two nodes i and j that are in the vertex cover.

Intuition

- Consider any node i in the vertex cover S.
- Since i is *tight*, its cost is exactly covered by the edges that are incident to it.
- However, an edge (i, j) might be incident to two nodes i and j that are in the vertex cover.
  - We will overcharge this edge.

Intuition

- Consider any node i in the vertex cover S.
- Since i is *tight*, its cost is exactly covered by the edges that are incident to it.
- However, an edge (i, j) might be incident to two nodes i and j that are in the vertex cover.
  - We will overcharge this edge.
  - But we only overcharge it by a factor of 2.

Second lemma

- Lemma: The set S and the prices p returned by the **Vertex-Cover-Approx** algorithm satisfy the following inequality:

$$w(S) \leq 2 \sum_{e \in E} p_e$$

Proof of the second lemma

Proof of the second lemma

- Consider a node i in S.

Proof of the second lemma

- Consider a node i in S.
  - This means that i is *tight*.

Proof of the second lemma

- Consider a node i in S.
  - This means that i is *tight*.
  - This means that $\sum_{e=(i,j)} p_e = w_i$

Proof of the second lemma

- Consider a node i in S.
  - This means that i is *tight*.
  - This means that $\sum_{e=(i,j)} p_e = w_i$
- Summing up over all nodes i in S: $w(S) = \sum_{i \in S} = \sum_{i \in S} \sum_{e=(i,j)} p_e$

Proof of the second lemma

- Consider a node i in S.
  - This means that i is *tight*.
  - This means that $\sum_{e=(i,j)} p_e = w_i$
- Summing up over all nodes i in S: $w(S) = \sum_{i \in S} = \sum_{i \in S} \sum_{e=(i,j)} p_e$
- An edge e=(i, j) can be included at most twice in this.

Proof of the second lemma

- Consider a node i in S.
  - This means that i is *tight*.
  - This means that $$\sum_{e=(i,j)} p_e = w_i$$
- Summing up over all nodes i in S: $$w(S) = \sum_{i \in S} = \sum_{i \in S} \sum_{e=(i,j)} p_e$$
- An edge e=(i, j) can be included at most twice in this.
- We get that

$$w(S) = \sum_{i \in S} \sum_{e=(i,j)} p_e \leq 2 \sum_{e \in E} p_e$$

From the two lemmas

$$\sum_{e \in E} p_e \leq w(S)$$

From the two lemmas

$$\sum_{e \in E} p_e \leq w(S)$$

$$w(S) \leq 2 \sum_{e \in E} p_e$$

Correctness

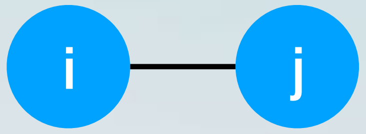
- First, S is a vertex cover.

Correctness

- First, S is a vertex cover.
  - Suppose that it is not.

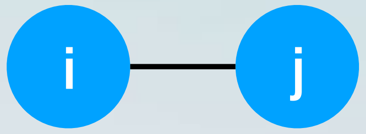
Correctness

- First, S is a vertex cover.
  - Suppose that it is not.
  - Then there is an edge e=(i, j) such that i and j are not in S.



Correctness

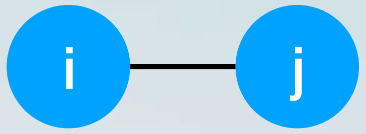
- First, S is a vertex cover.
  - Suppose that it is not.



  - Then there is an edge e=(i, j) such that i and j are not in S.
  - This means that neither i or j are *tight*.

Correctness

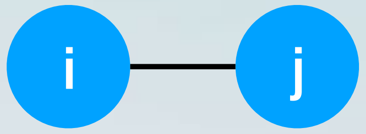
- First, S is a vertex cover.
  - Suppose that it is not.



  - Then there is an edge e=(i, j) such that i and j are not in S.
  - This means that neither i or j are *tight*.
  - But then why did the algorithm terminate?

Correctness

- First, S is a vertex cover.
  - Suppose that it is not.



  - Then there is an edge e=(i, j) such that i and j are not in S.
  - This means that neither i or j are *tight*.
  - But then why did the algorithm terminate?

"While there is an edge e=(i, j) such that neither i nor j is tight
Select such an edge e"

Approximation Ratio

- Let S^* be the minimum weight vertex cover.
- Recall that S is the vertex cover returned by the algorithm.

From the two lemmas

$$\sum_{e \in E} p_e \leq w(S)$$

$$w(S) \leq 2 \sum_{e \in E} p_e$$

This holds for any vertex cover **S**, also for S*

Approximation Ratio

- Let S^* be the minimum weight vertex cover.
- Recall that S is the vertex cover returned by the algorithm.

Approximation Ratio

- Let S* be the minimum weight vertex cover.
- Recall that S is the vertex cover returned by the algorithm.
- We have:

$$\sum_{e \in E} p_e \leq w(S^*) \qquad\qquad w(S) \leq 2 \sum_{e \in E} p_e$$

Approximation Ratio

- Let S* be the minimum weight vertex cover.
- Recall that S is the vertex cover returned by the algorithm.
- We have:

$$\sum_{e \in E} p_e \leq w(S^*) \qquad w(S) \leq 2 \sum_{e \in E} p_e$$

- Which clearly implies: $w(S) \leq 2W(S^*)$

Vertex Cover as an ILP

Minimise $\sum_{i \in V} x_i w_i$

subject to $x_i + x_j \geq 1,$ **for all** $(i, j) \in E$

$x_i \geq 0,$ **for all** $i \in V$

$x_i \in \{0, 1\},$ **for all** $i \in V$

Vertex Cover as an ILP

Minimise $\sum_{i \in V} x_i w_i$

For each edge, one of the endpoints has to be in the vertex cover.

subject to $x_i + x_j \geq 1,$ **for all** $(i, j) \in E$

$x_i \geq 0,$ **for all** $i \in V$

$x_i \in \{0, 1\},$ **for all** $i \in V$

Vertex Cover LP-relaxation

Minimise $\sum_{i \in V} x_i w_i$

subject to $x_i + x_j \geq 1,$ **for all** $(i, j) \in E$

$x_i \geq 0,$ **for all** $i \in V$

Vertex Cover LP-relaxation

Minimise $\sum_{i \in V} x_i w_i$

subject to $x_i + x_j \geq 1,$ **for all** $(i, j) \in E$

Possible fractional values, e.g., $x_i \geq 0,$ **for all** $i \in V$

x_i = *0.3*, x_j =*0.7*

The Dual

Minimise $\sum_{i \in V} x_i w_i$ **Primal**

subject to $x_i + x_j \geq 1,$ **for all** $(i, j) \in E$

$x_i \geq 0,$ **for all** $i \in V$

Maximise $\sum_{e \in E} p_e$ **Dual**

subject to $\sum_{e=(i,j)} p_e \leq w_i,$ **for all** $i \in V$

$p_e \geq 0,$ **for all** $e \in E$

The Dual

Minimise $\sum_{i \in V} x_i w_i$ **Primal**

subject to $x_i + x_j \geq 1,$ **for all** $(i, j) \in E$

$x_i \geq 0,$ **for all** $i \in V$

Maximise $\sum_{e \in E} p_e$ **Dual**

subject to $\sum_{e=(i,j)} p_e \leq w_i,$ **for all** $i \in V$

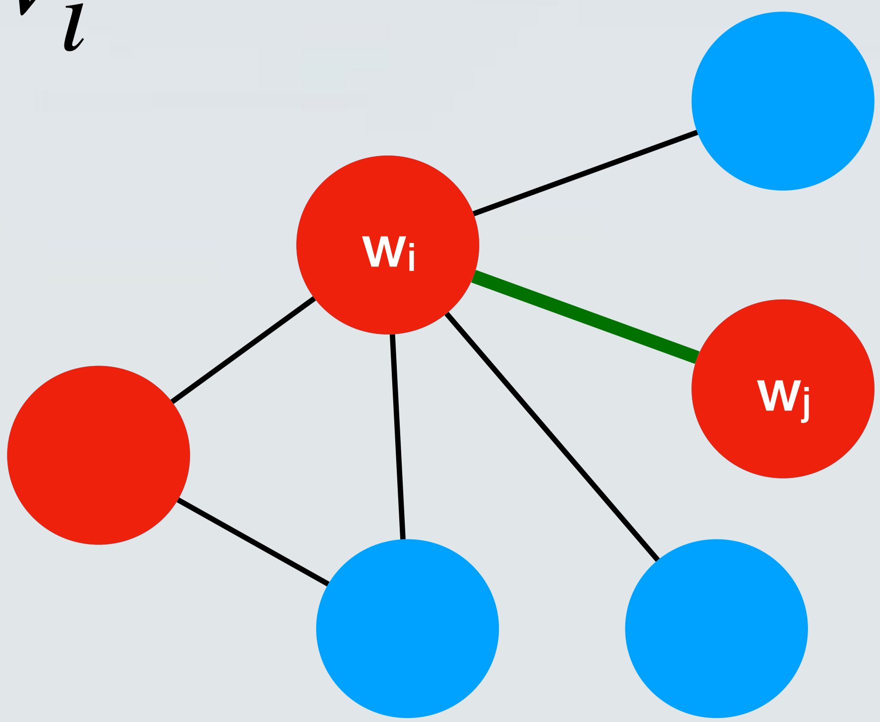
$p_e \geq 0,$ **for all** $e \in E$

Wait a sec,
I've seen
this before.

Fair Pricing

- Given a vertex i, we never ask the edges that "*use it*" to pay more than the cost of the vertex.

$$\sum_{e=(i,j)} p_e \leq w_i$$



What are we really doing?

- We are taking a variable p_e of the *dual* of the LP-relaxation.
- We increase the value of this variable, until some constraint becomes *tight*.

$$\sum_{e=(i,j)} p_e \leq w_i, \quad \textbf{for all } i \in V$$

$\mathbf{p_e}$

What are we really doing?

- We are taking a variable p_e of the *dual* of the LP-relaxation.

- We increase the value of this variable, until some constraint becomes *tight*.

$$\sum_{e=(i,j)} p_e = w_i, \quad \textbf{for all } i \in V$$

What are we really doing?

- We are taking a variable p_e of the *dual* of the LP-relaxation.

- We increase the value of this variable, until some constraint becomes *tight*.

What are we really doing?

- We are taking a variable p_e of the *dual* of the LP-relaxation.
- We increase the value of this variable, until some constraint becomes *tight*.
- Because constraints of the *dual* correspond to variables of the *primal*, there is an associated variable x_i of the *primal*.

What are we really doing?

- We are taking a variable p_e of the *dual* of the LP-relaxation.
- We increase the value of this variable, until some constraint becomes *tight*.
- Because constraints of the *dual* correspond to variables of the *primal*, there is an associated variable x_i of the *primal*.
- We set the value of that variable in the *primal* to *1*.

Primal-dual method

- We start with an infeasible integral solution x to the *primal* and a feasible fractional solution y to the *dual*.
- We increase the value of some y_j until some constraint (that contains y_j) becomes *tight*.
  - We obtain a better feasible fractional solution y to the *dual*.
  - We increase the corresponding variable x_i of *the primal* to obtain a still infeasible integral solution x to the primal, which however violates fewer constraints.
- We end up with a feasible integral solution x to the primal, and a feasible fractional solution y to *the dual*.
- We compare the two solutions.

The Primal-dual method