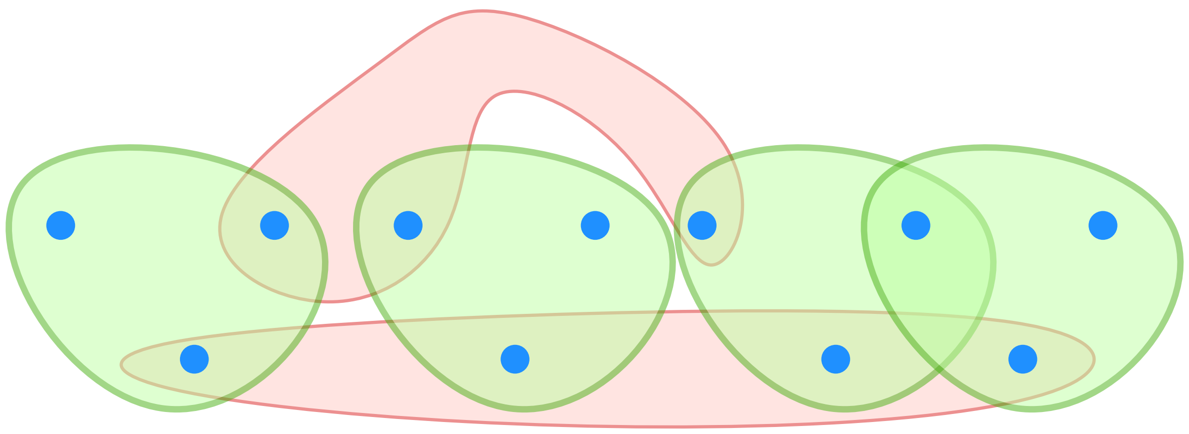


SetCover

greedy approximation algorithm

layering algorithm for weighted VertexCover

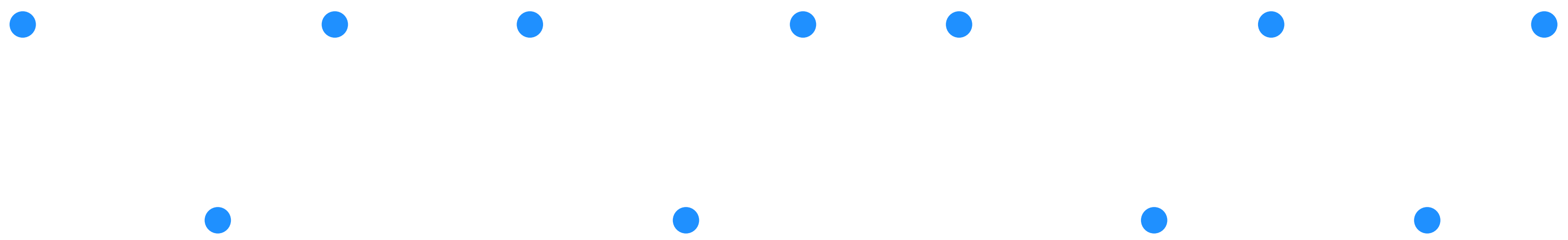
application to ShortestSuperString

SetCover (cardinality)

Let U be some ground set (universe),

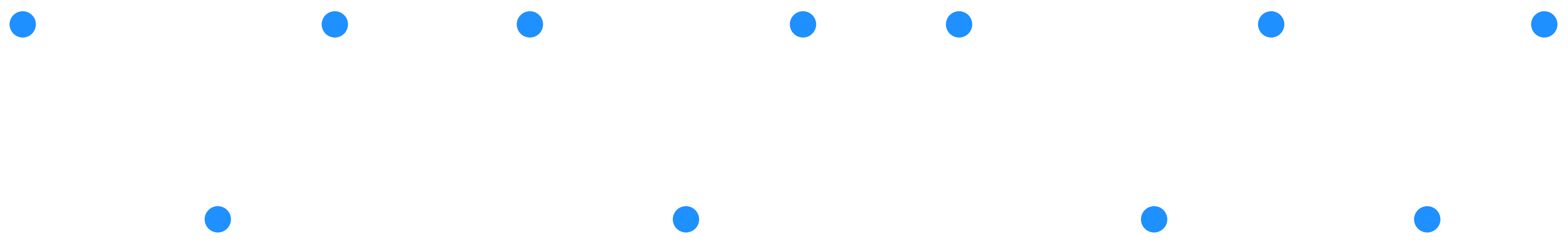
SetCover (cardinality)

Let U be some ground set (universe),



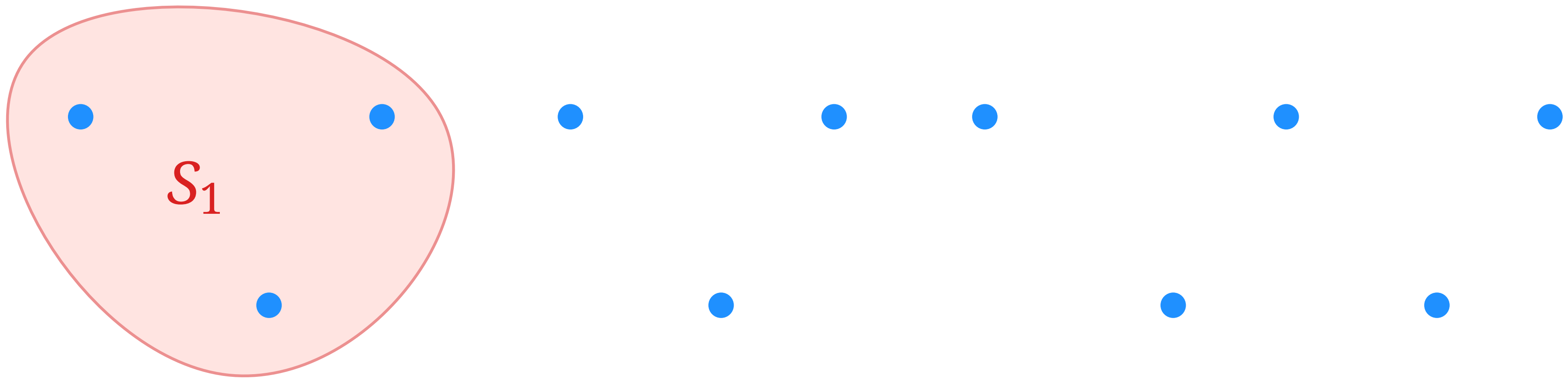
SetCover (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.



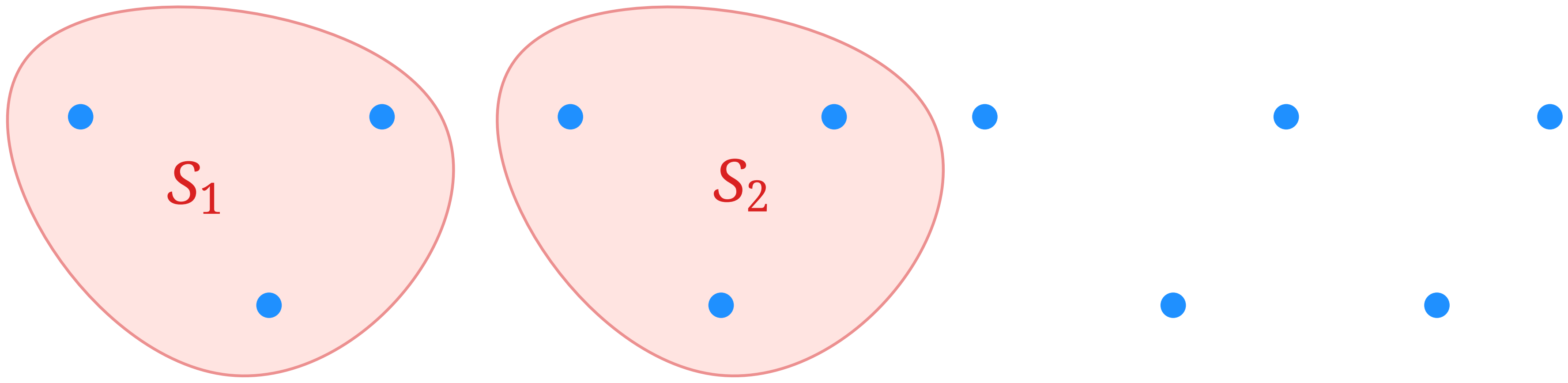
SetCover (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.



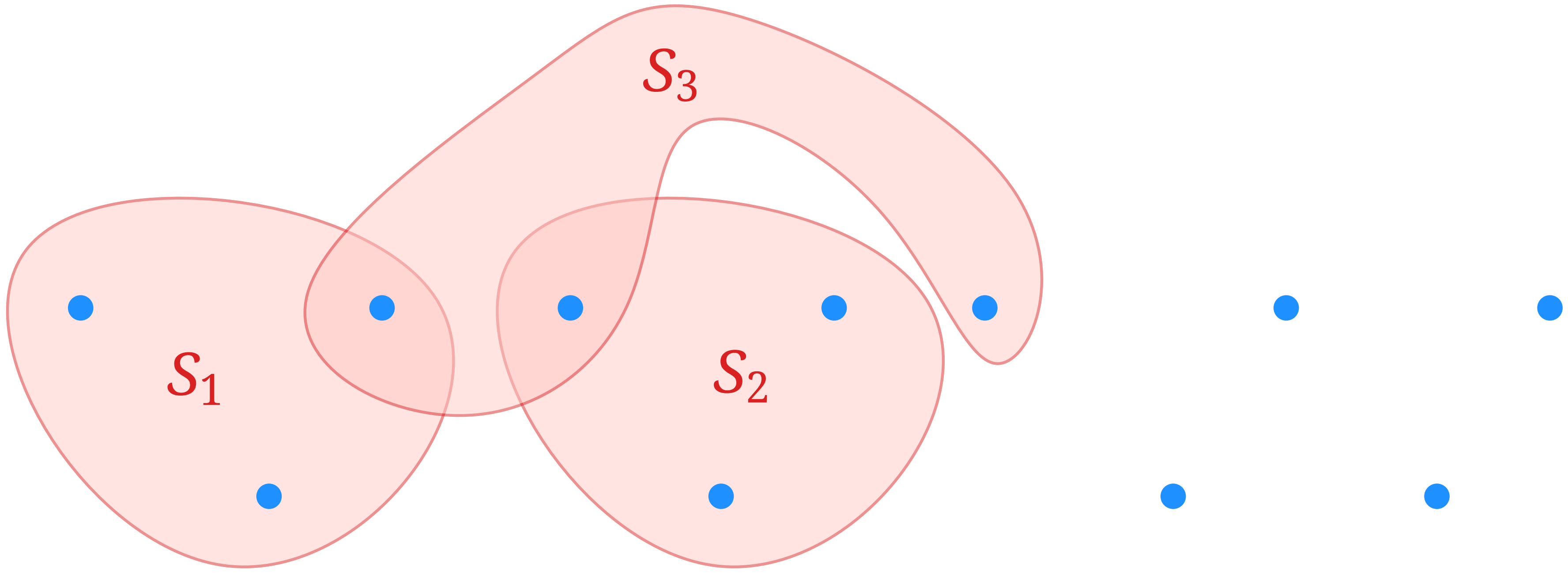
SetCover (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.



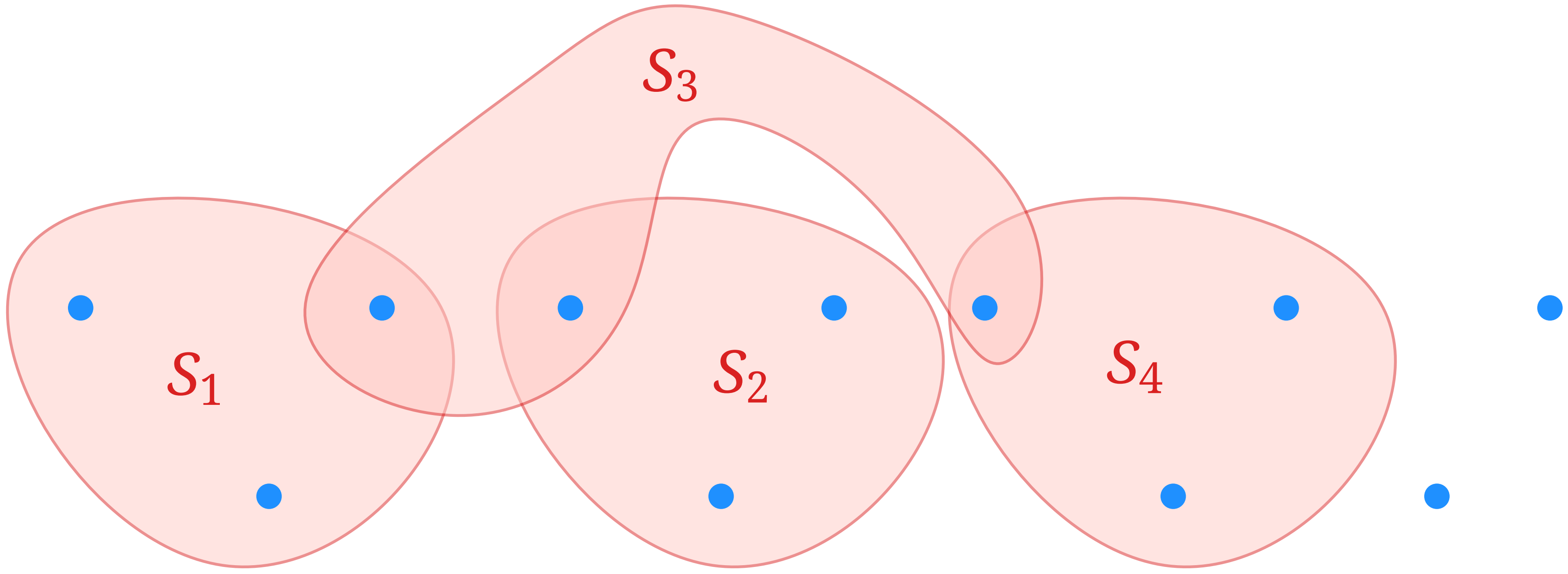
SetCover (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.



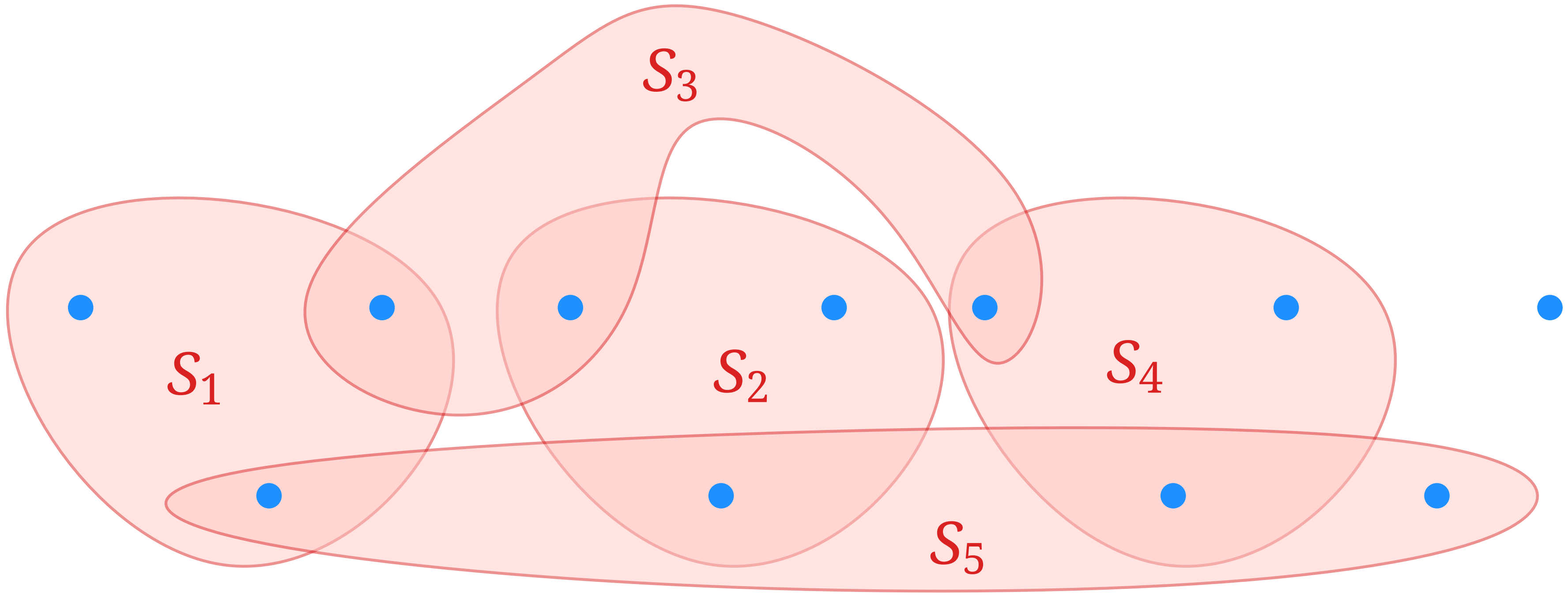
SetCover (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.



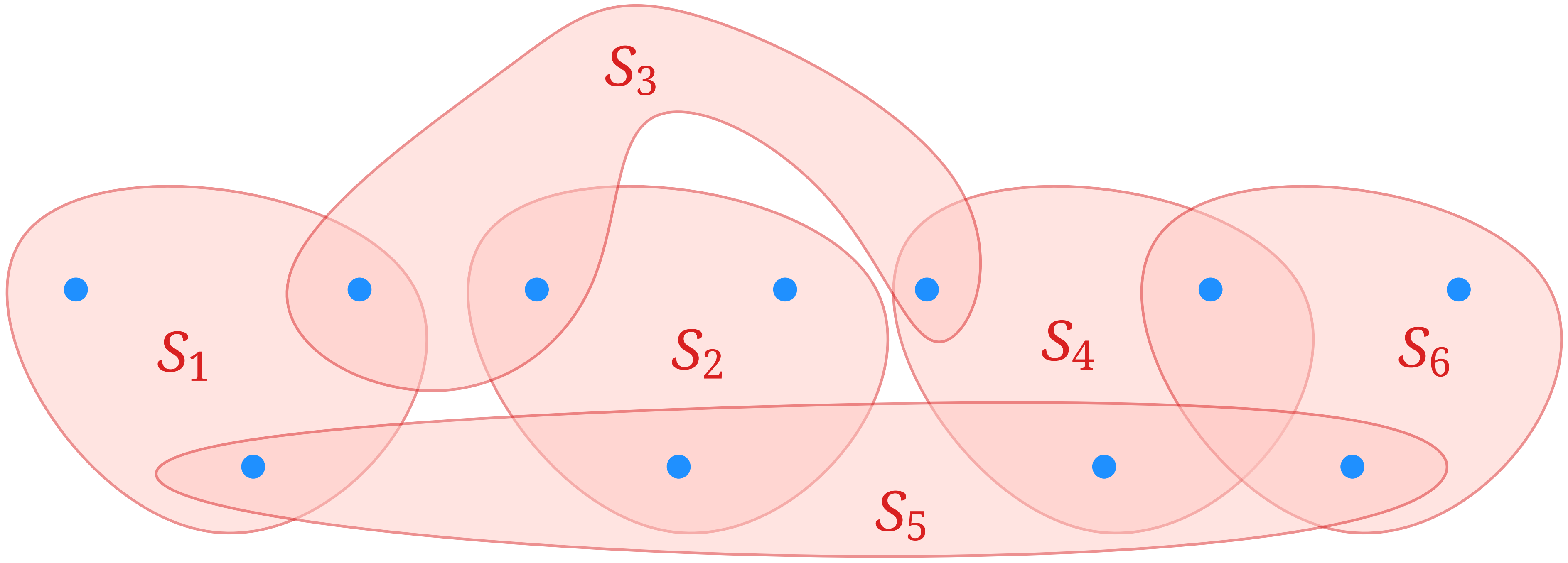
SetCover (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.



SetCover (cardinality)

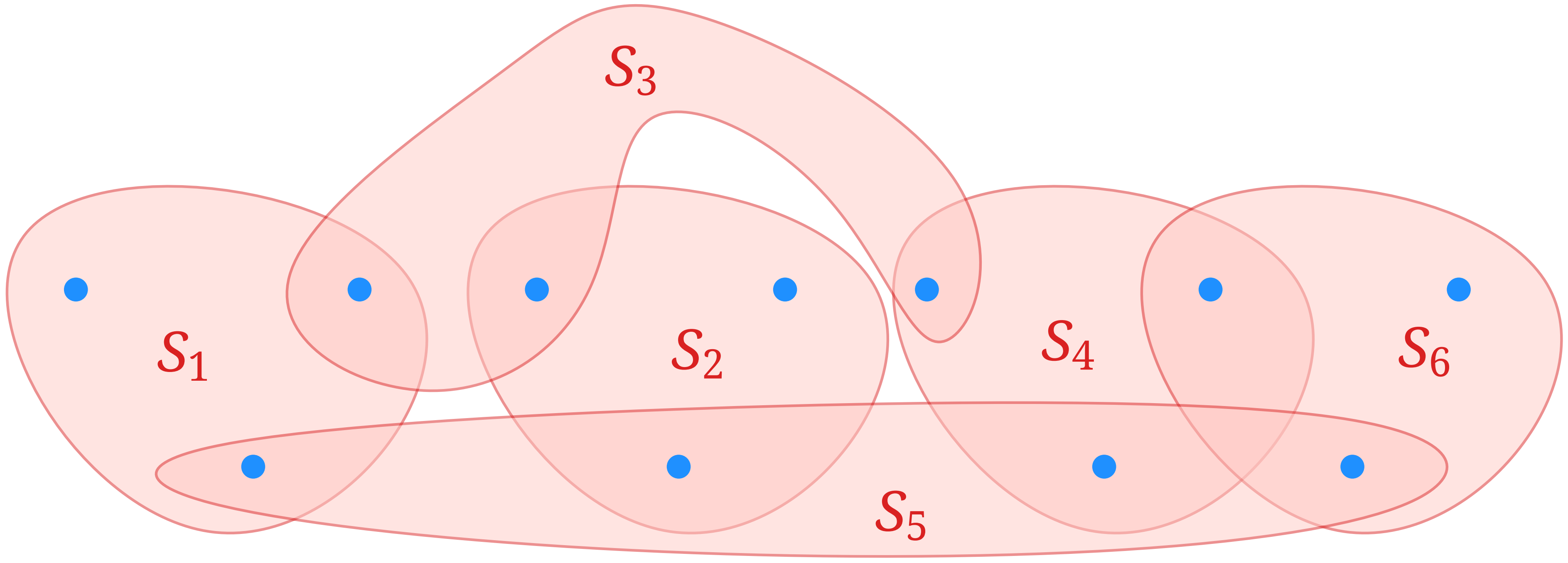
Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.



SETCOVER (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.

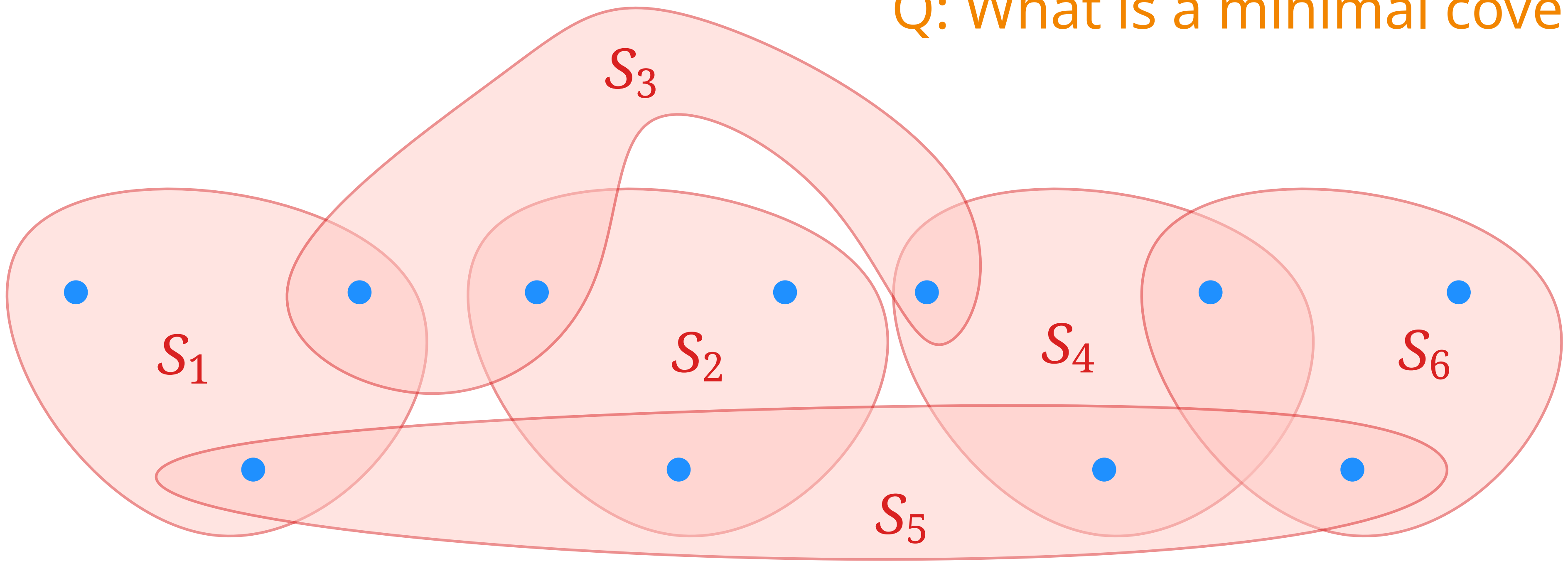


SetCover (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.

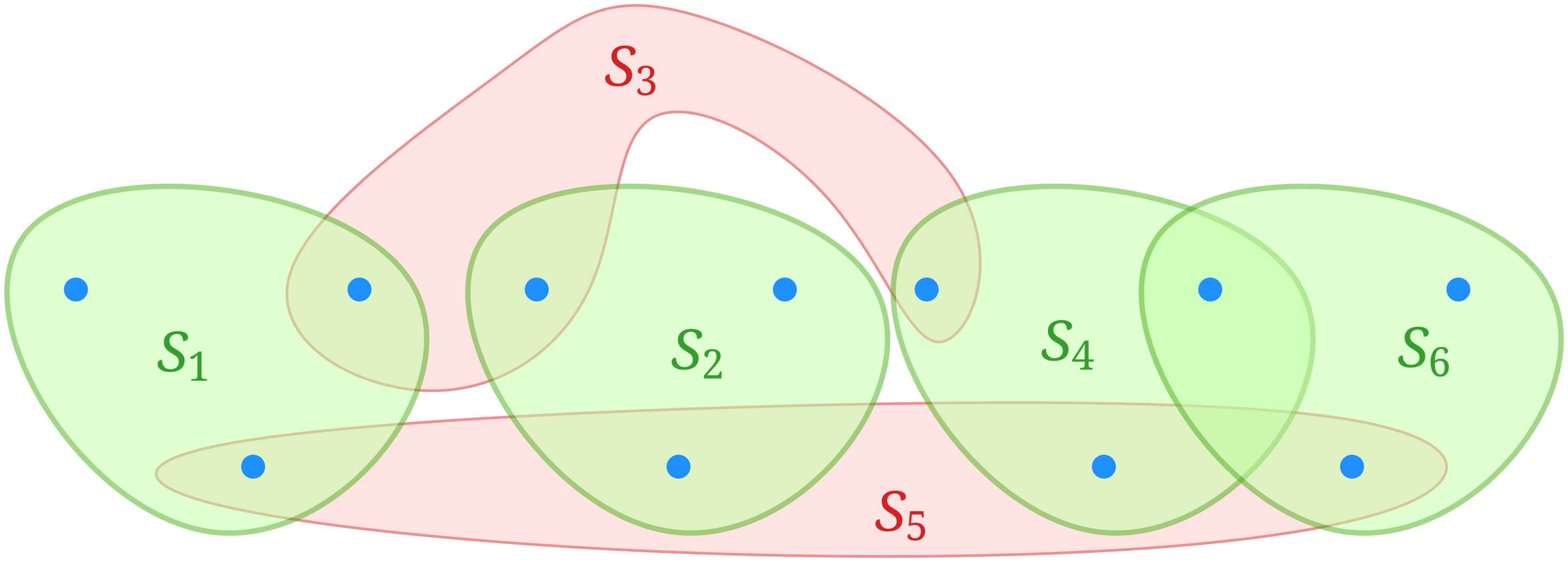
Q: What is a minimal cover here?



SetCover (cardinality)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

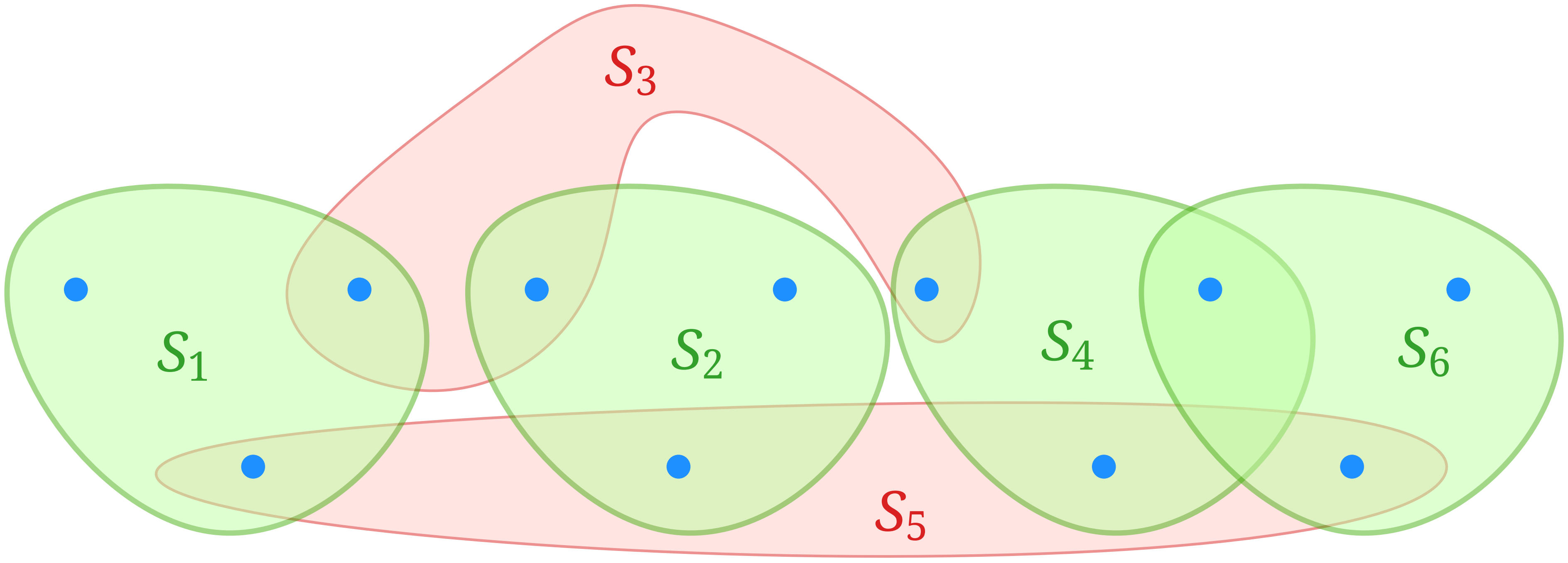
Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.



SetCover (general)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.

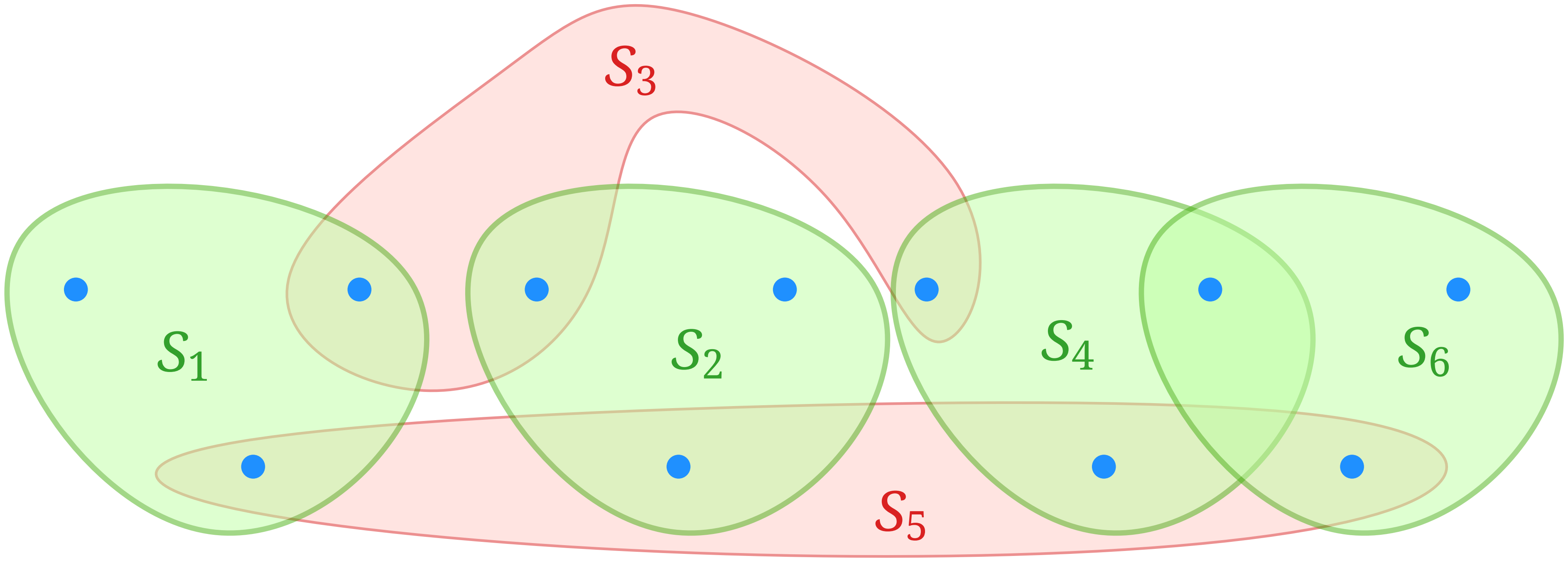


SetCover (general)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum cardinality.

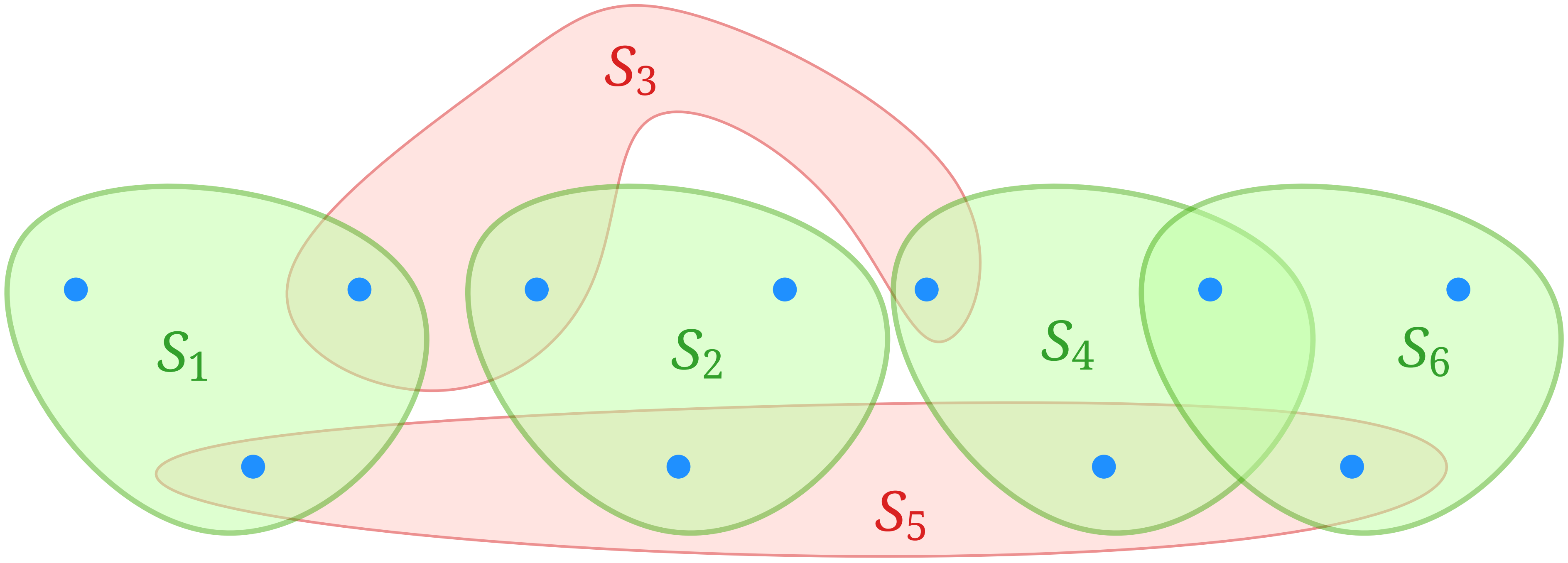


SetCover (general)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum ~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.

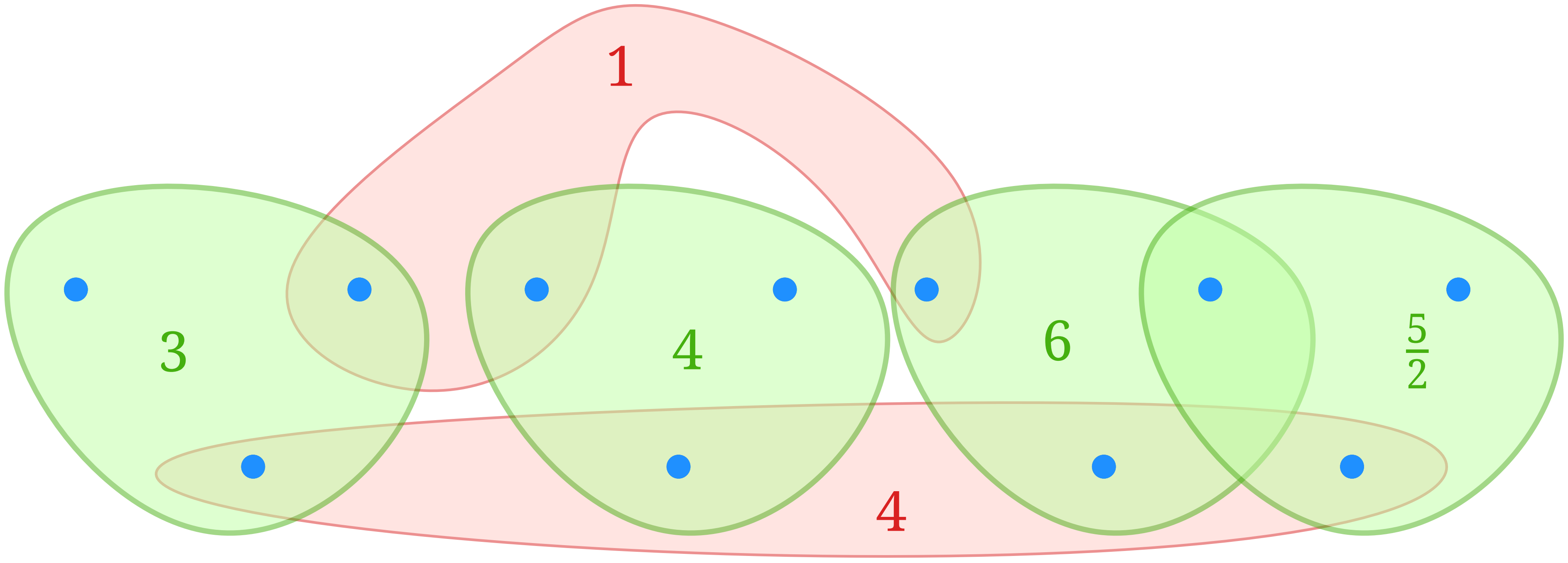


SetCover (general)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a cover $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum ~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.



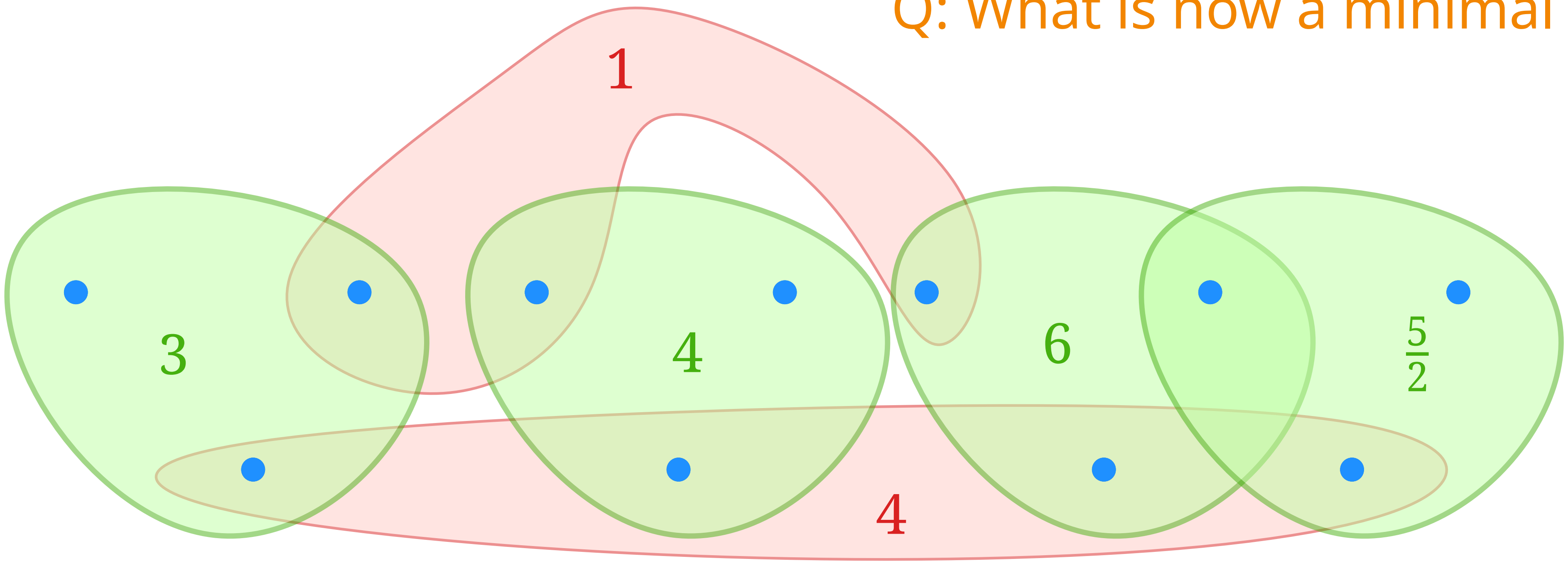
SetCover (general)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum
~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.

Q: What is now a minimal cover?



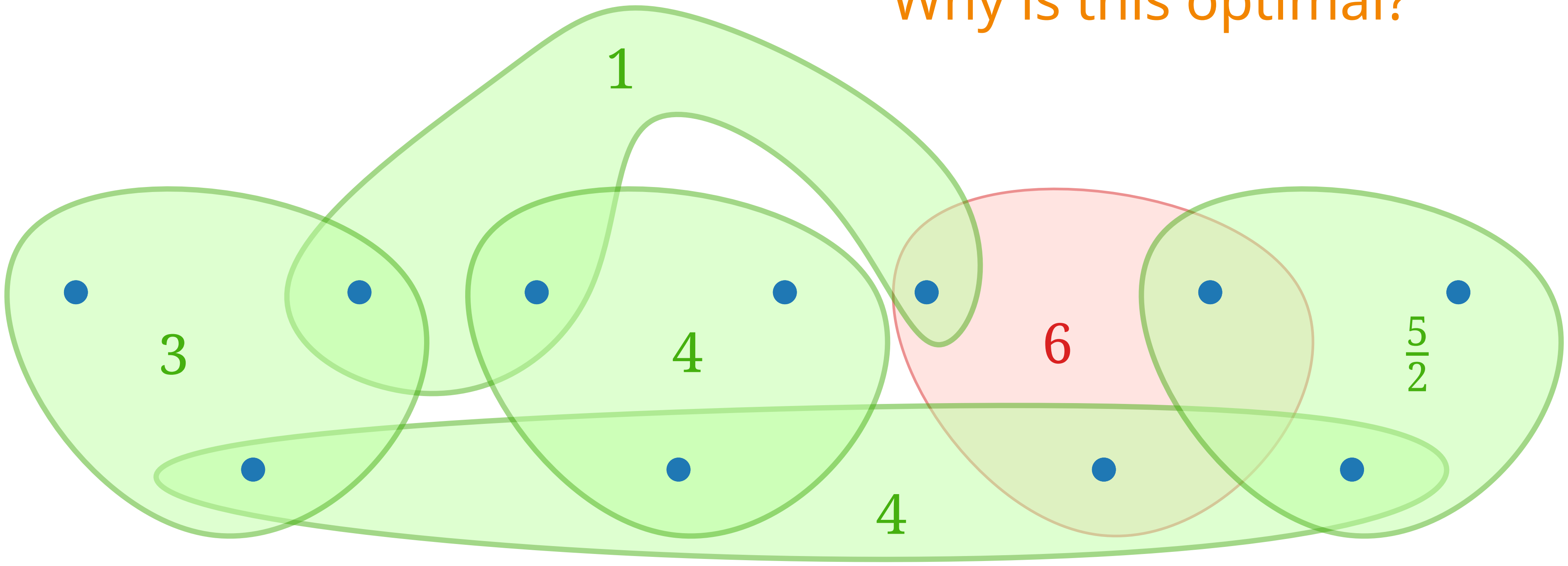
SetCover (general)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum ~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.

Why is this optimal?



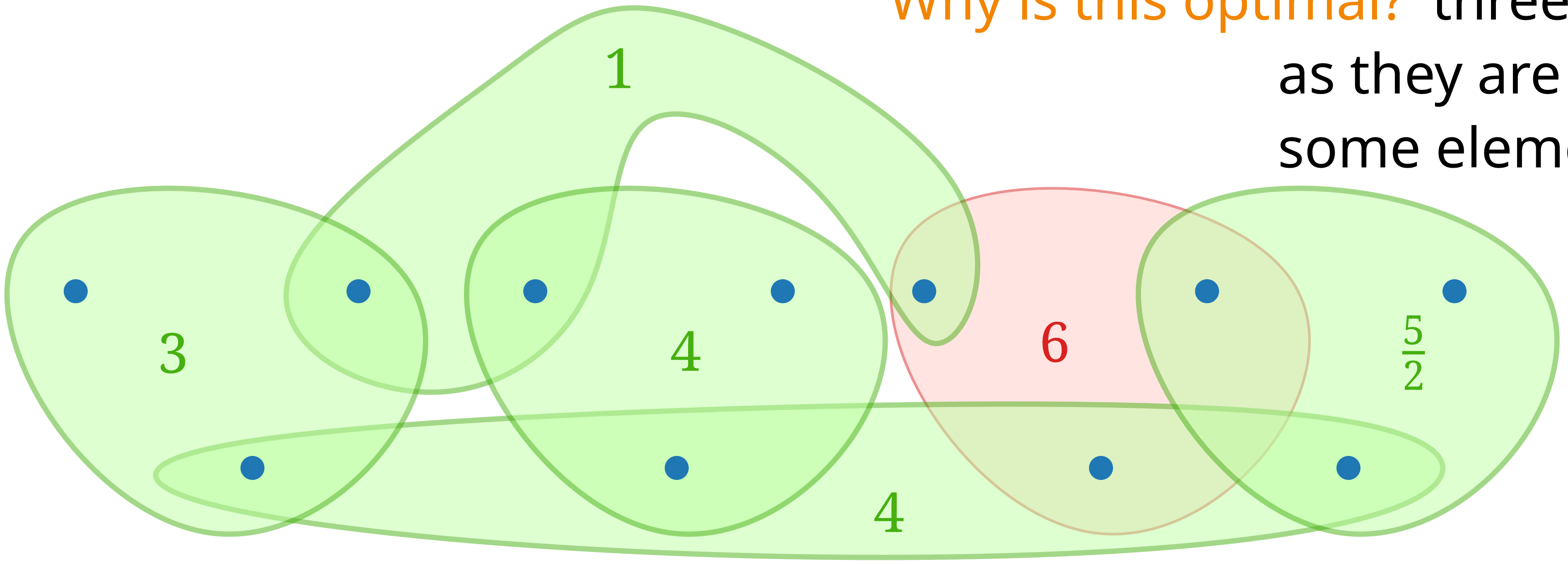
SetCover (general)

Let U be some ground set (universe),
and let $\mathcal{S}$ be a family of subsets of U with $\bigcup \mathcal{S} = U$.

Each $S \in \mathcal{S}$ has cost $c(S) > 0$.

Find a **cover** $\mathcal{S}' \subseteq \mathcal{S}$ of U (i.e., with $\bigcup \mathcal{S}' = U$) of minimum ~~cardinality.~~ total cost $c(\mathcal{S}') := \sum_{S \in \mathcal{S}'} c(S)$.

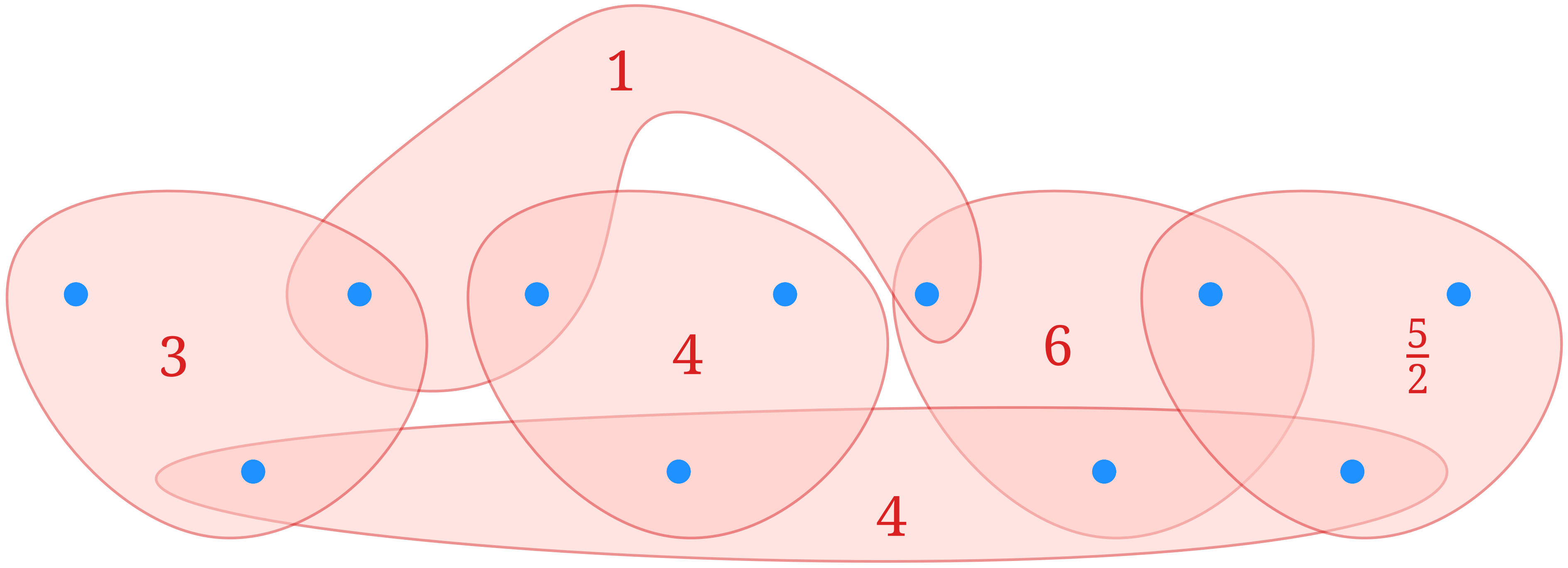
Why is this optimal? three sets need to be chosen as they are the only ones to cover some elements; then $1 + 4 < 6$



Greedy Approximation Algorithm for SetCover

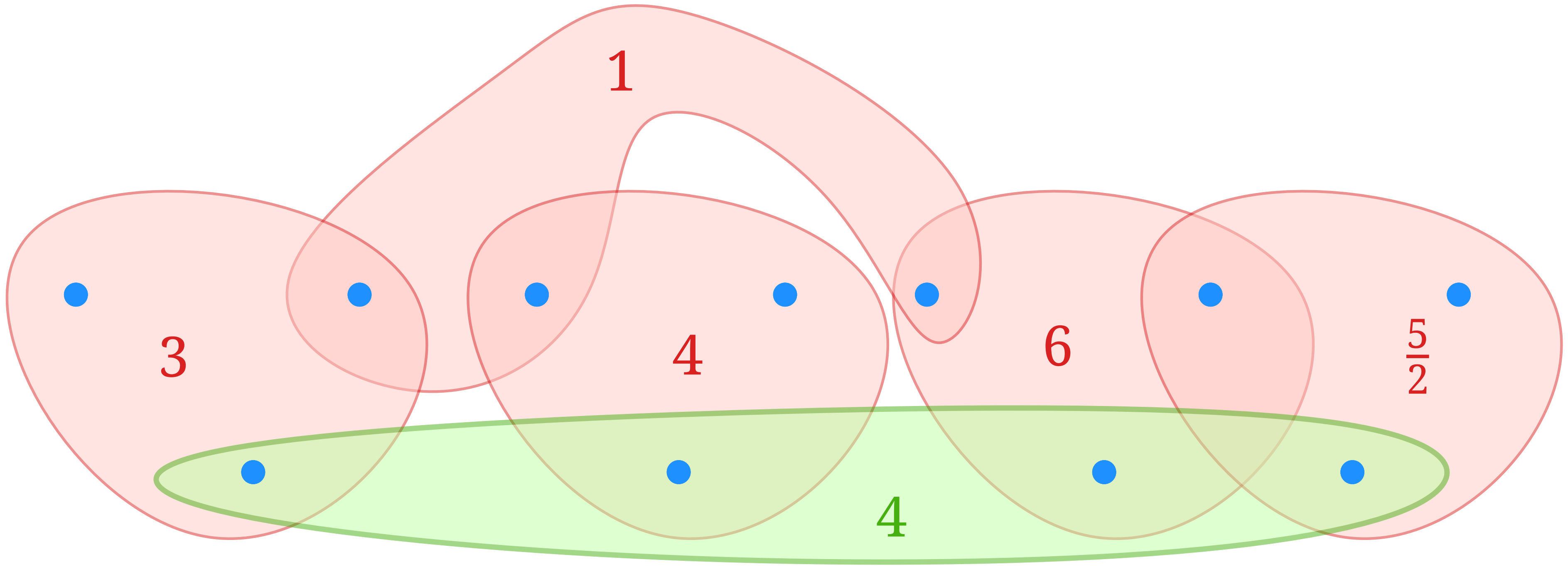
Iterative "Buying" of Elements

What is the real cost of picking a set?



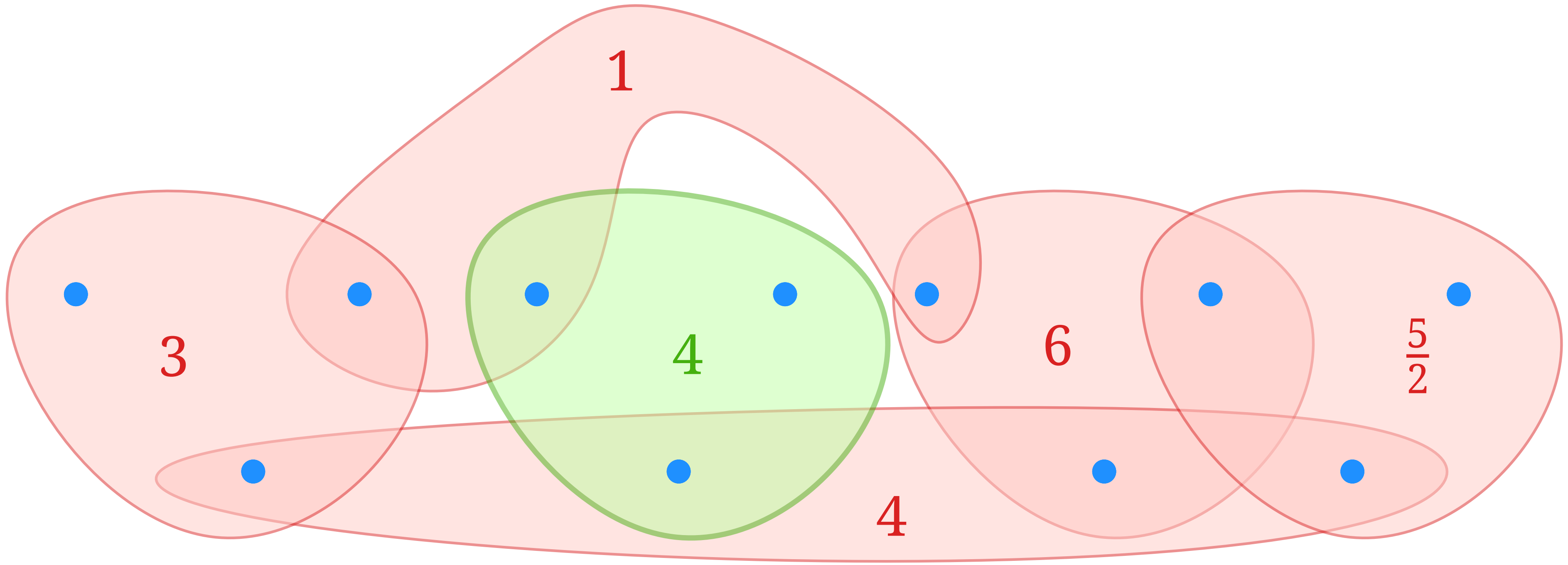
Iterative “Buying” of Elements

What is the real cost of picking a set?



Iterative “Buying” of Elements

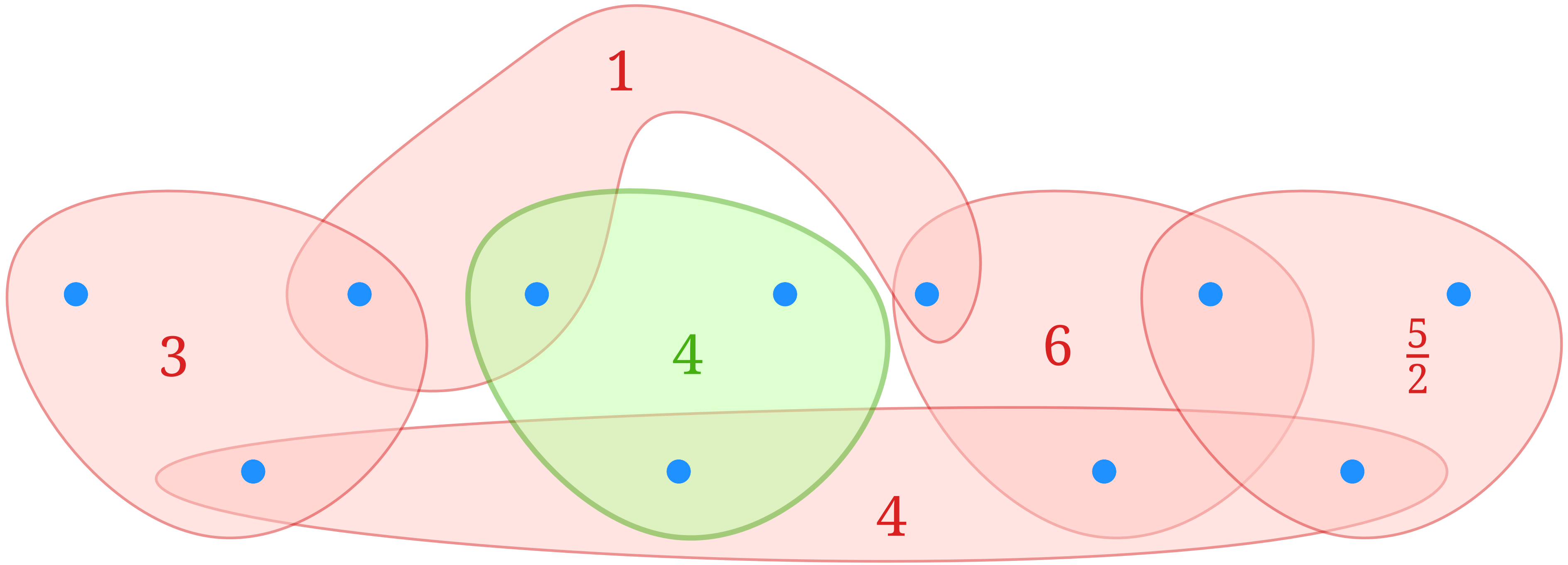
What is the real cost of picking a set?



Iterative "Buying" of Elements

What is the real cost of picking a set?

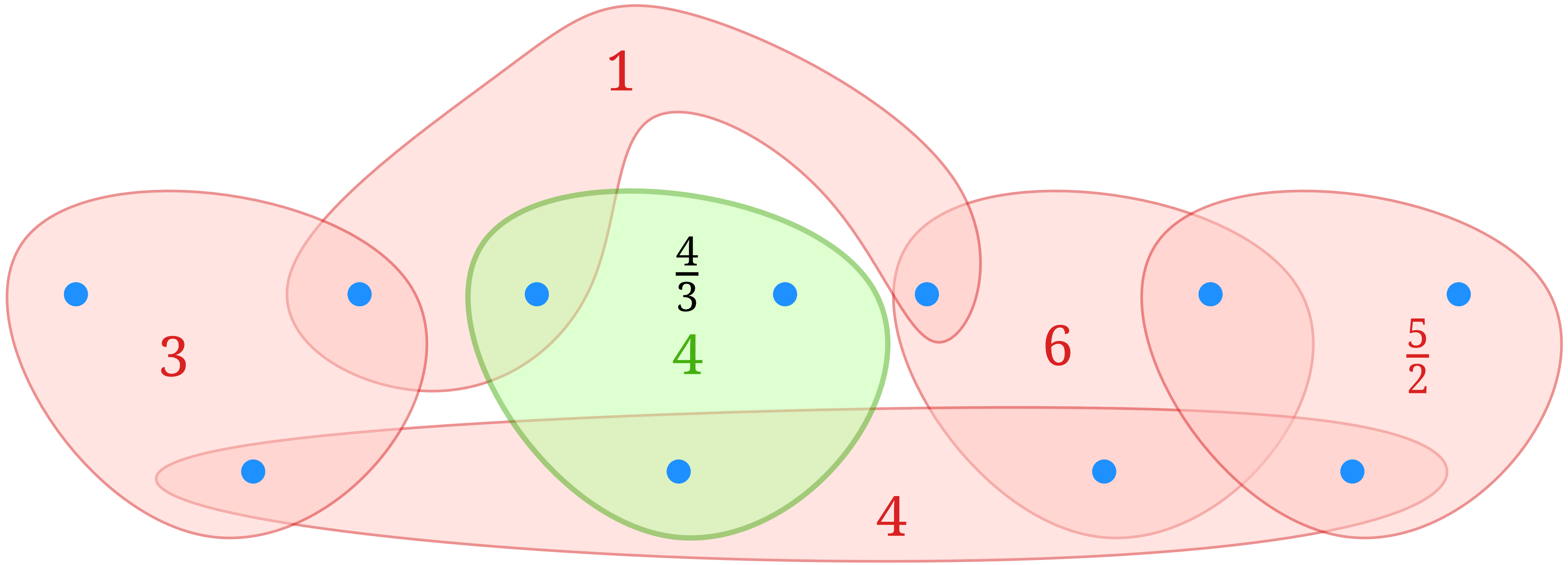
Set with k elements and cost c has per-element cost c/k.



Iterative "Buying" of Elements

What is the real cost of picking a set?

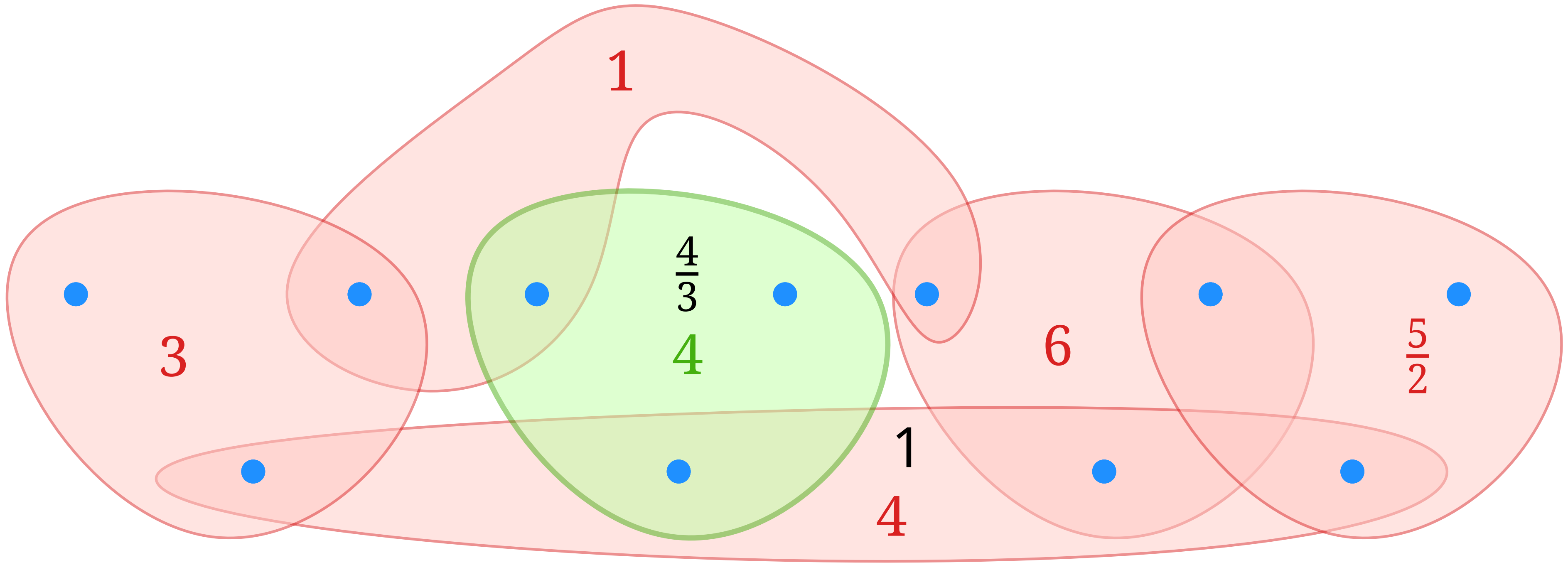
Set with k elements and cost c has per-element cost c/k.



Iterative "Buying" of Elements

What is the real cost of picking a set?

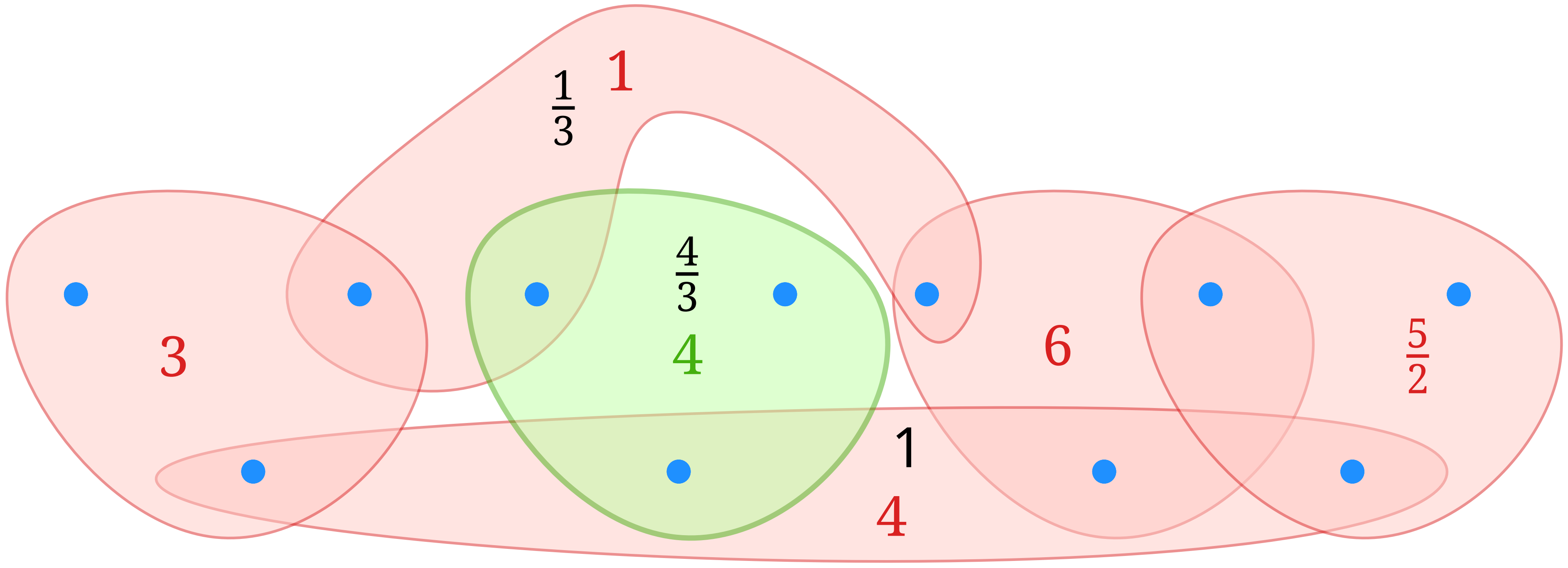
Set with k elements and cost c has per-element cost c/k.



Iterative "Buying" of Elements

What is the real cost of picking a set?

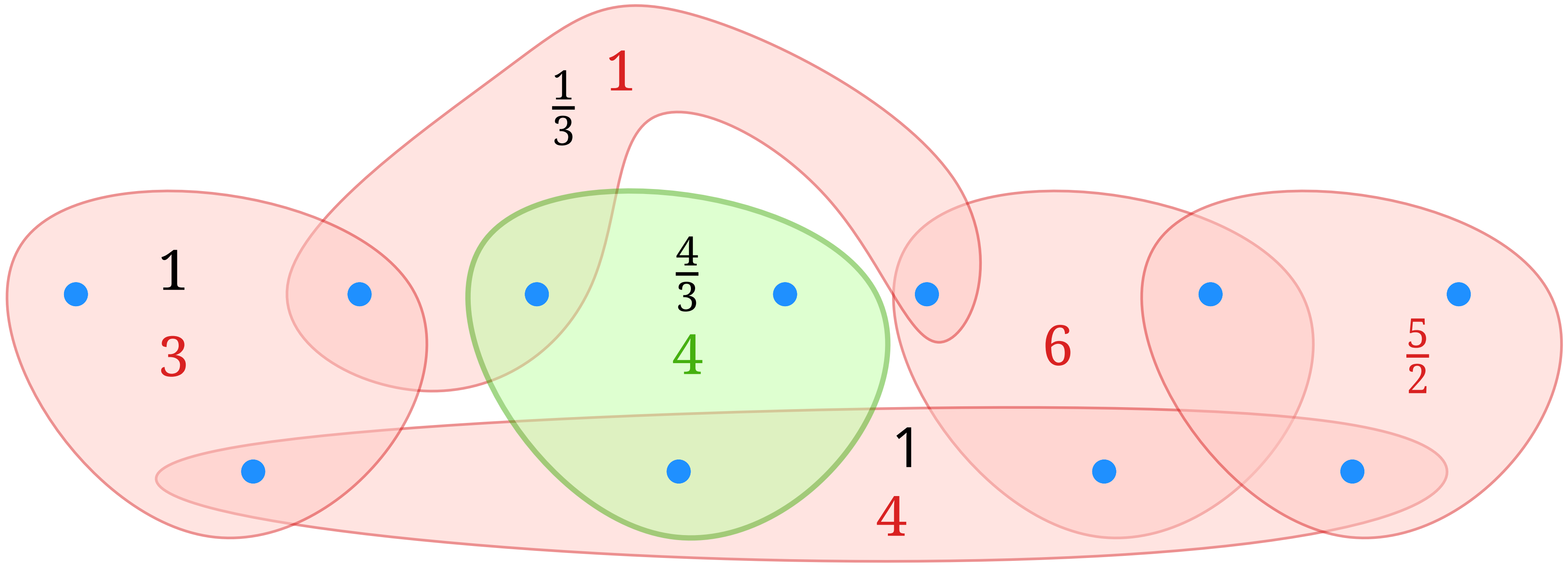
Set with k elements and cost c has per-element cost c/k.



Iterative "Buying" of Elements

What is the real cost of picking a set?

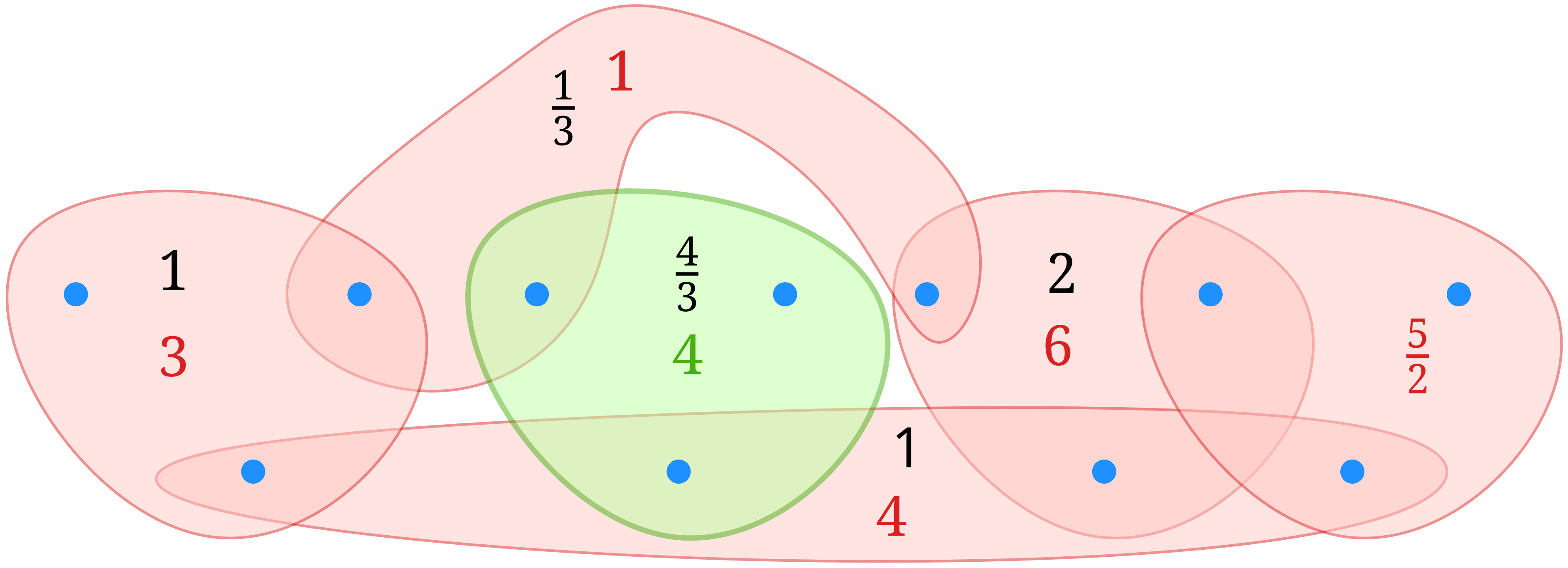
Set with k elements and cost c has per-element cost c/k.



Iterative “Buying” of Elements

What is the real cost of picking a set?

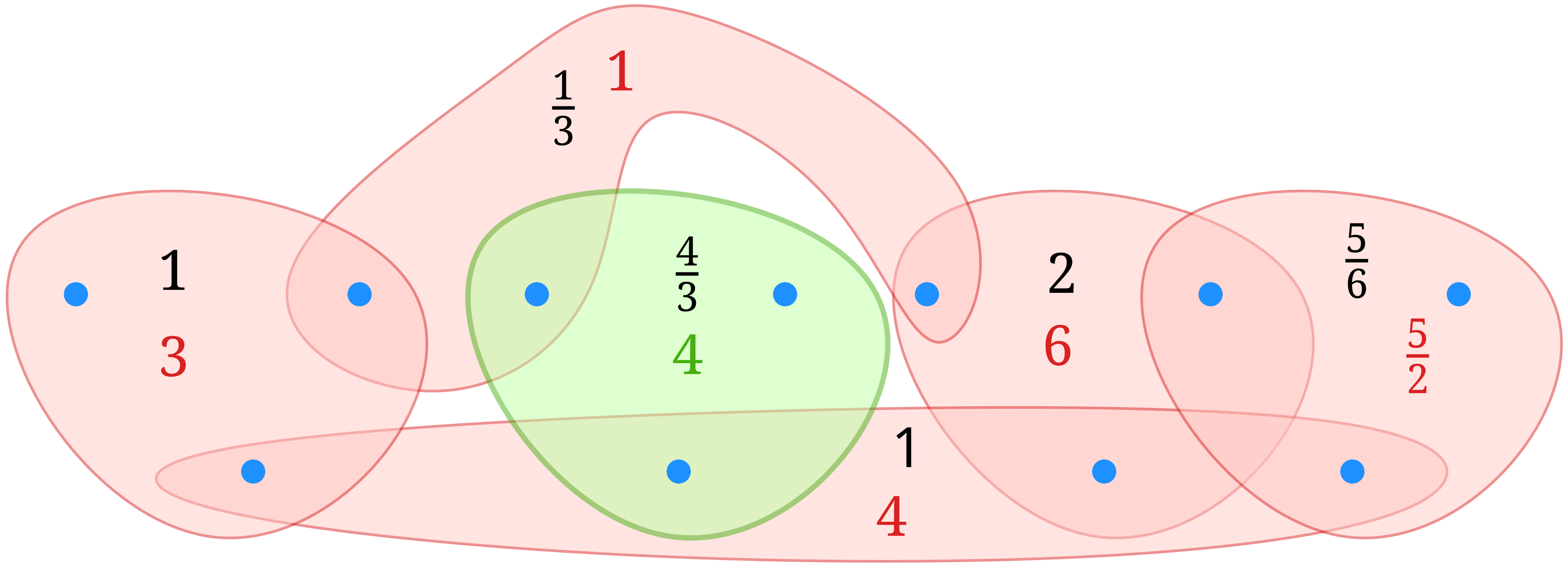
Set with k elements and cost c has per-element cost c/k.



Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

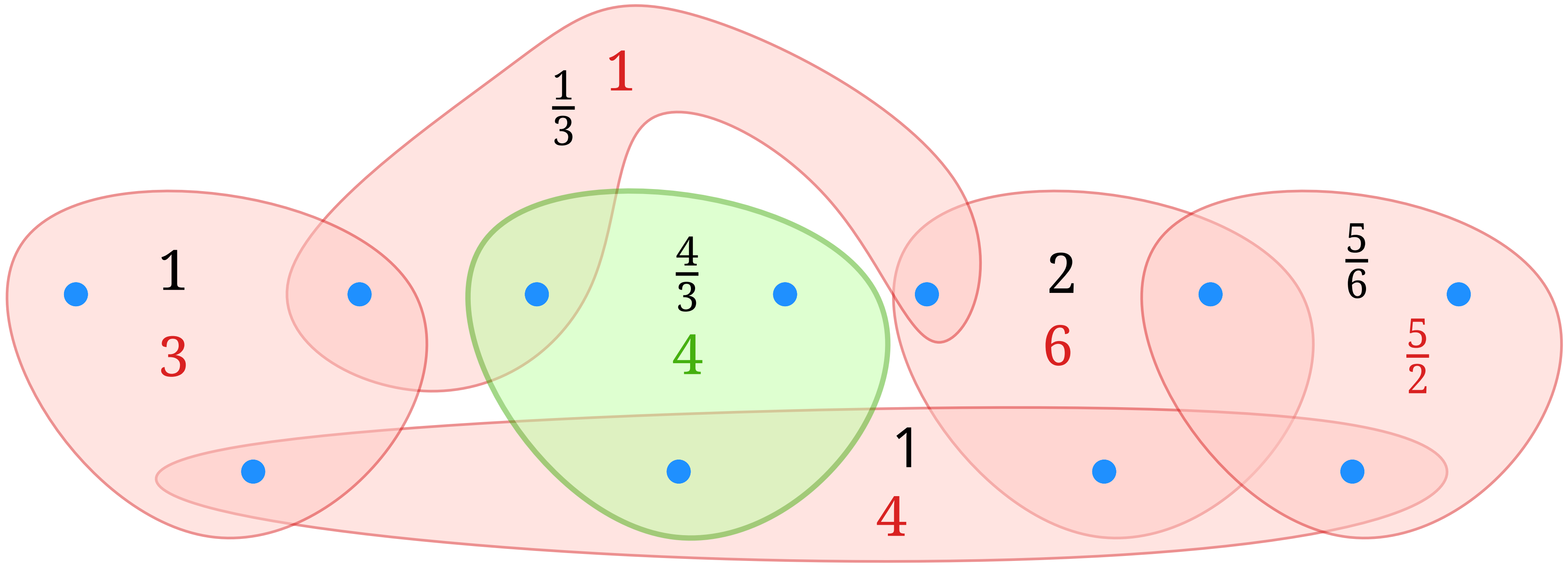


Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?



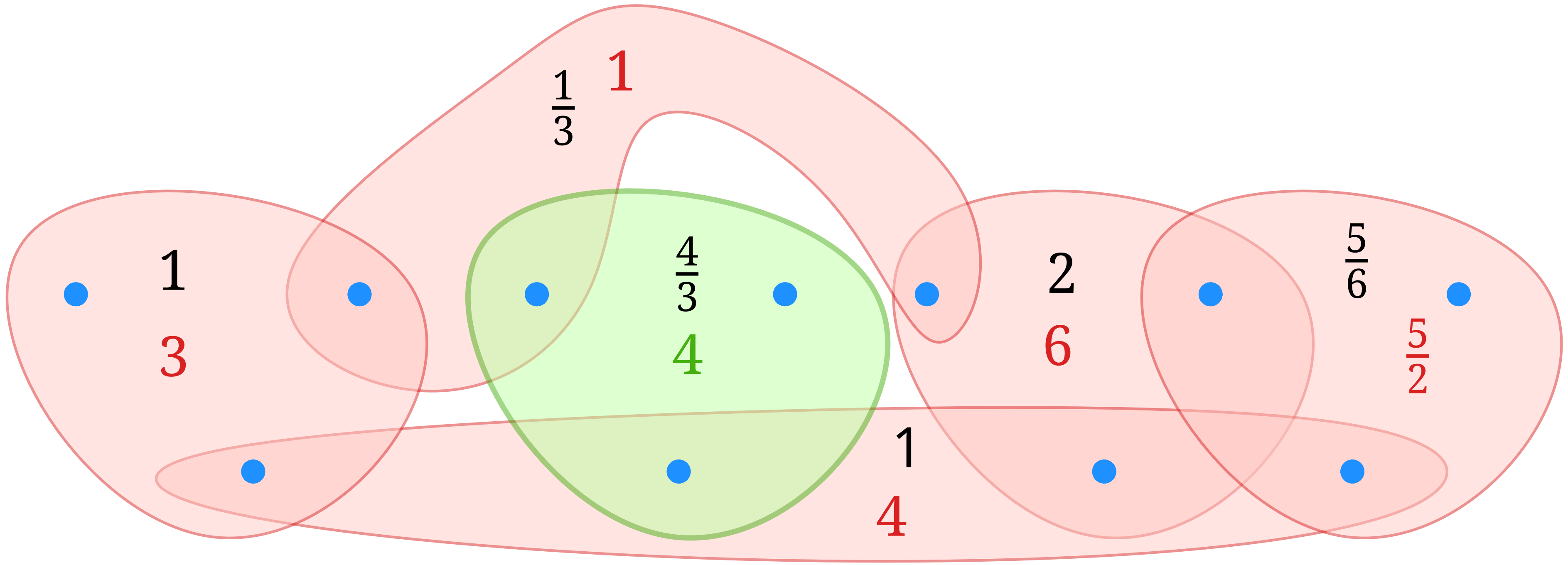
Iterative “Buying” of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we “buy” a set?

Fix price of elements bought and recompute per-element cost.



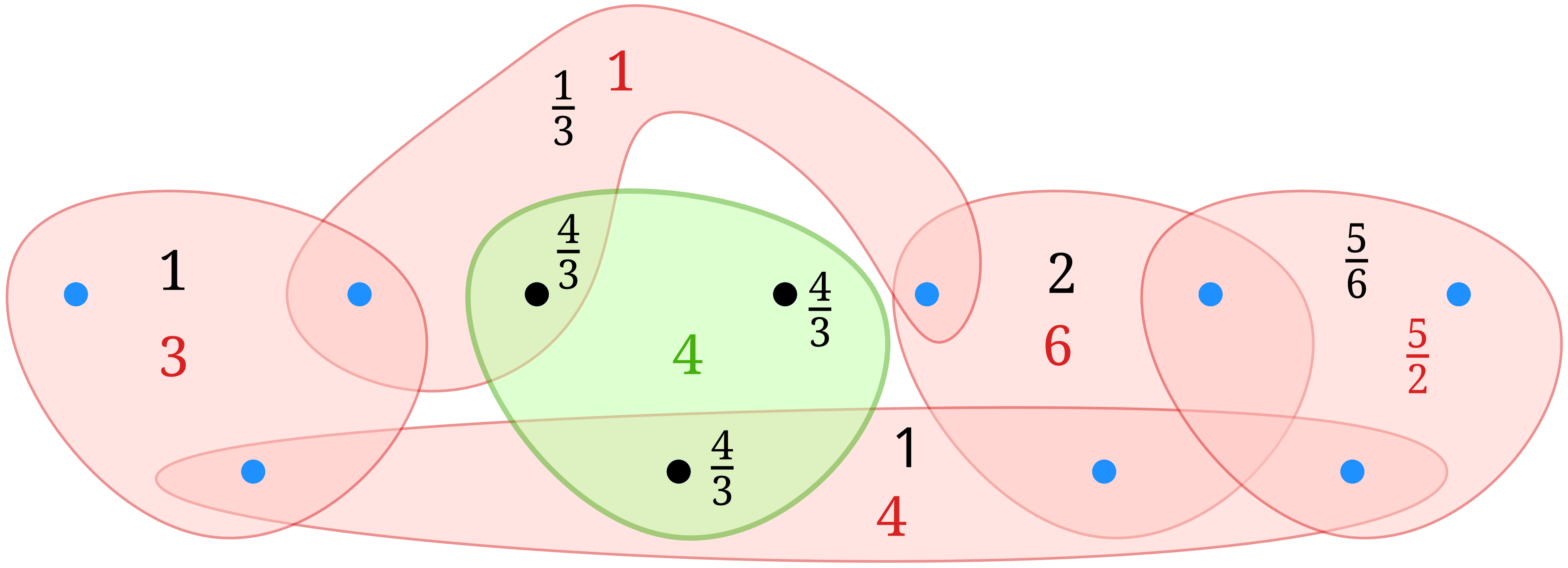
Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



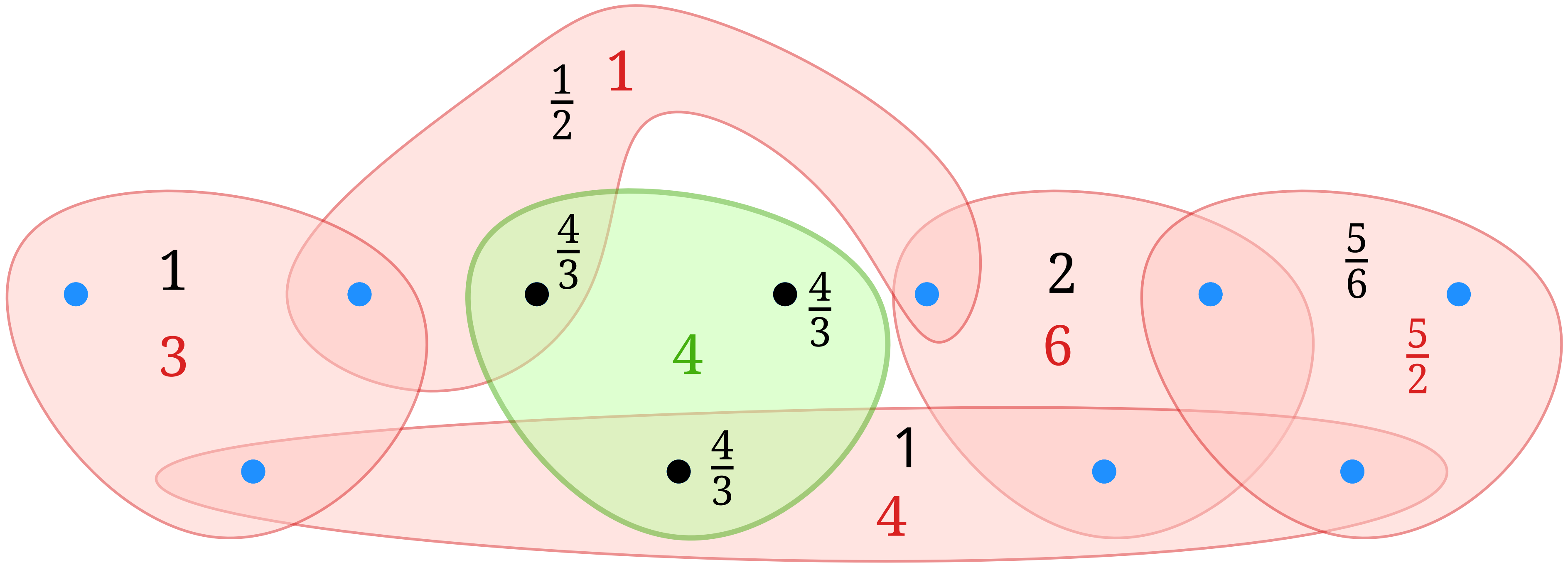
Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



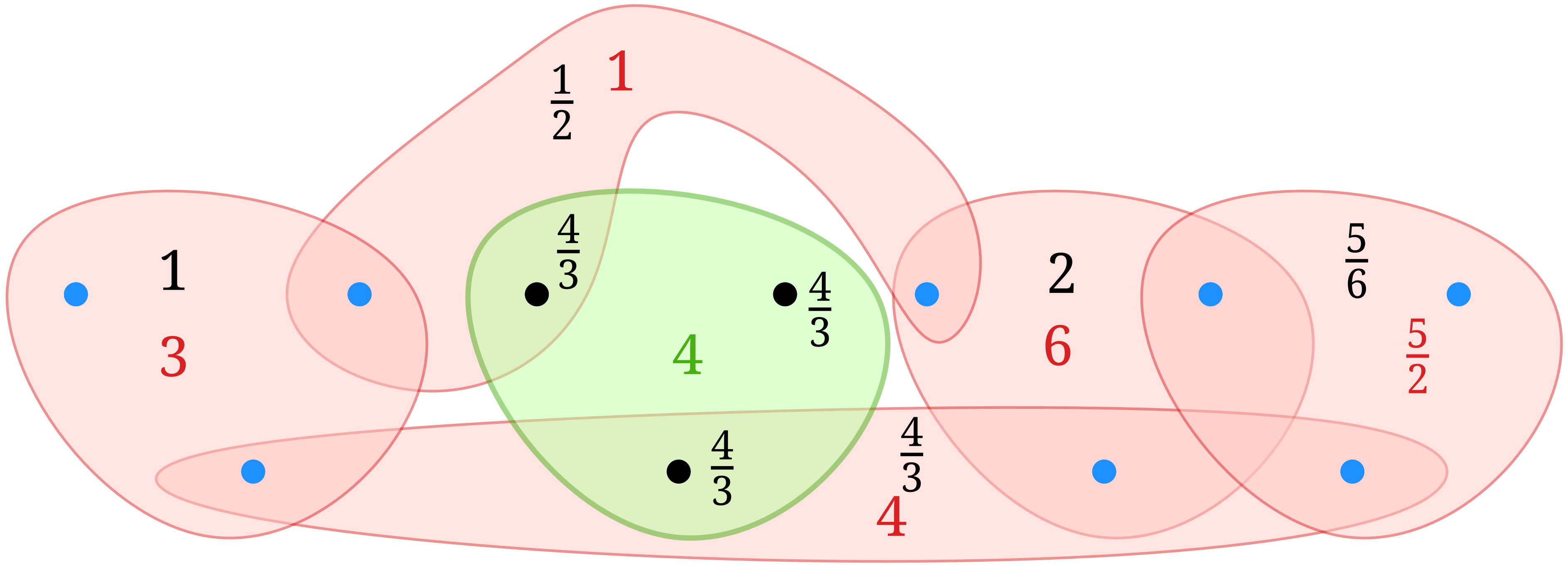
Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



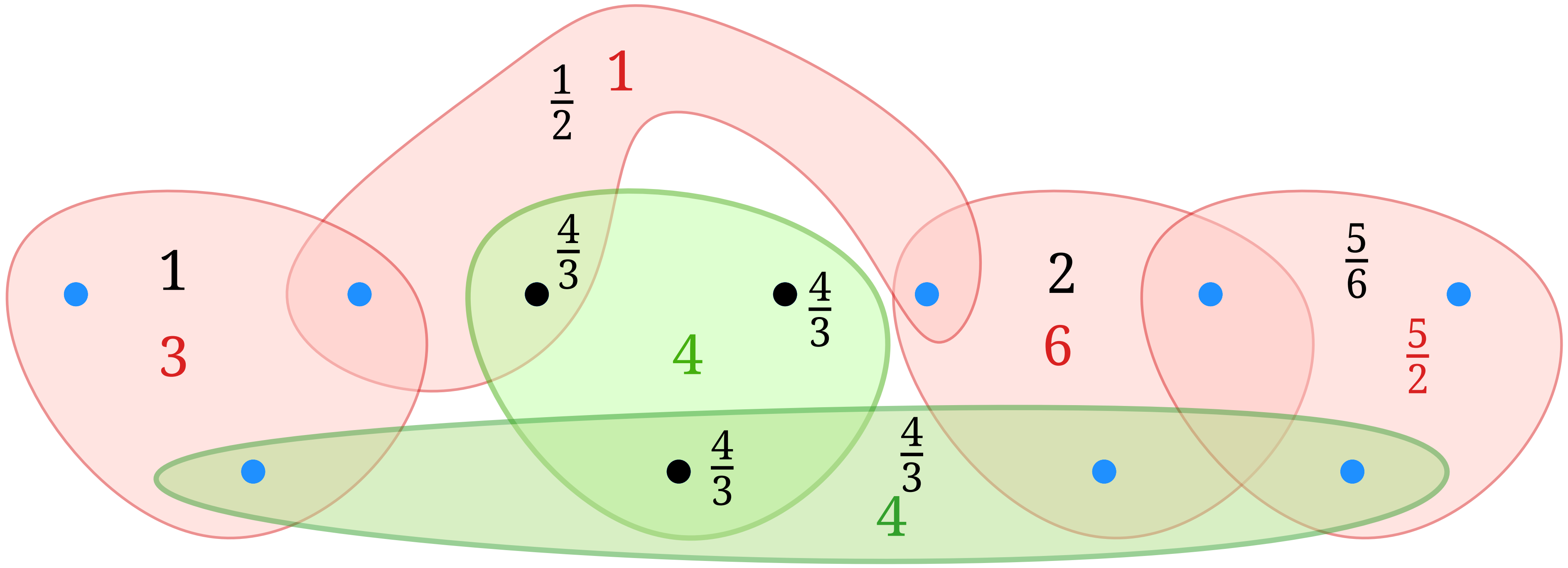
Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



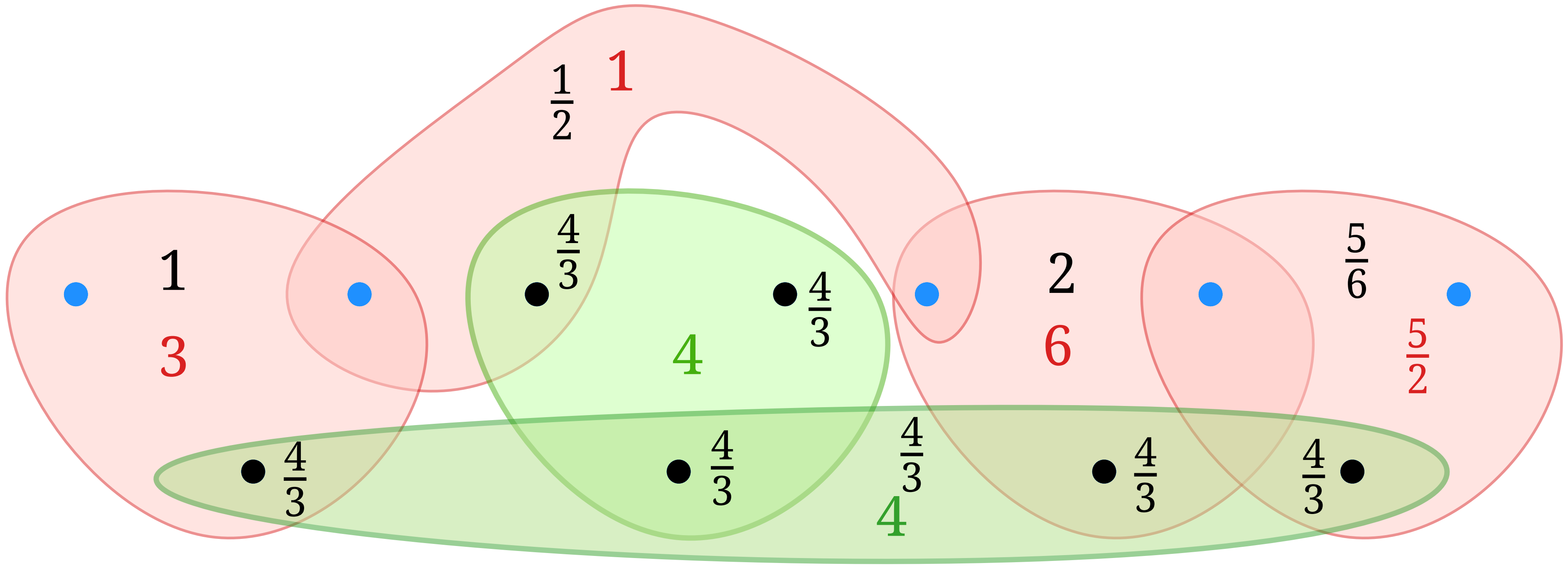
Iterative “Buying” of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we “buy” a set?

Fix price of elements bought and recompute per-element cost.



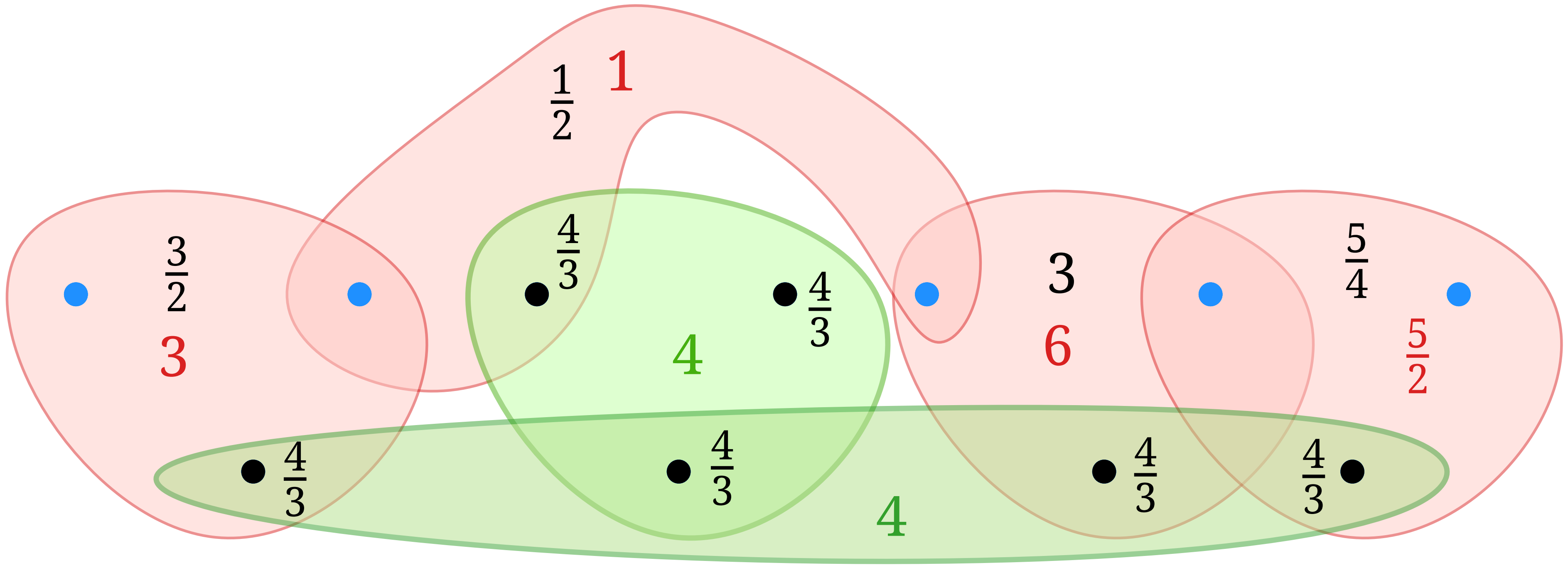
Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



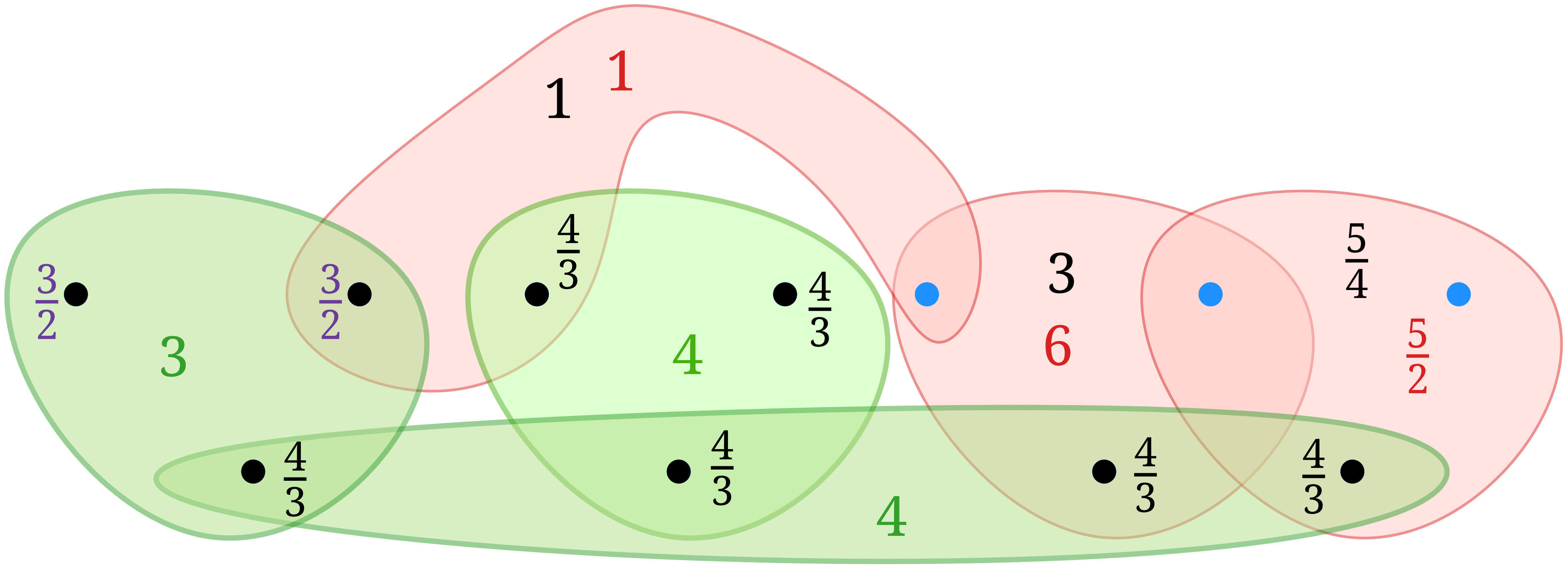
Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



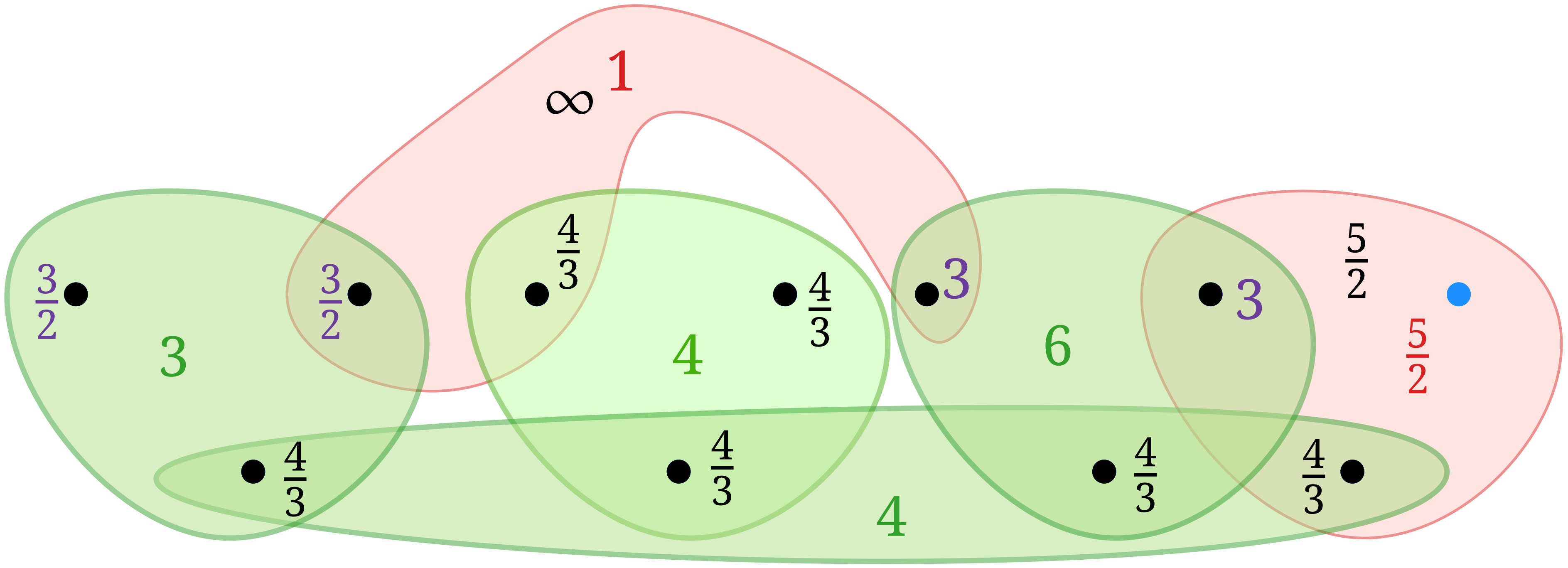
Iterative "Buying" of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.



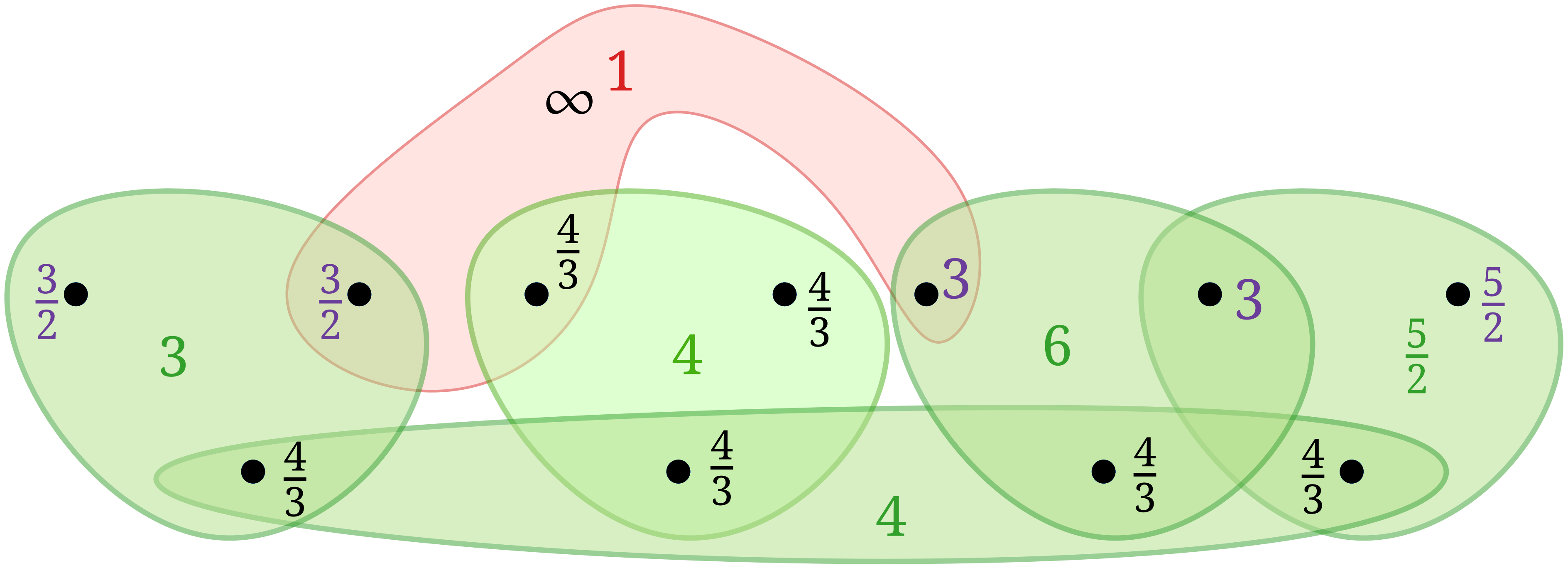
Iterative “Buying” of Elements

What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we “buy” a set?

Fix price of elements bought and recompute per-element cost.



Iterative “Buying” of Elements

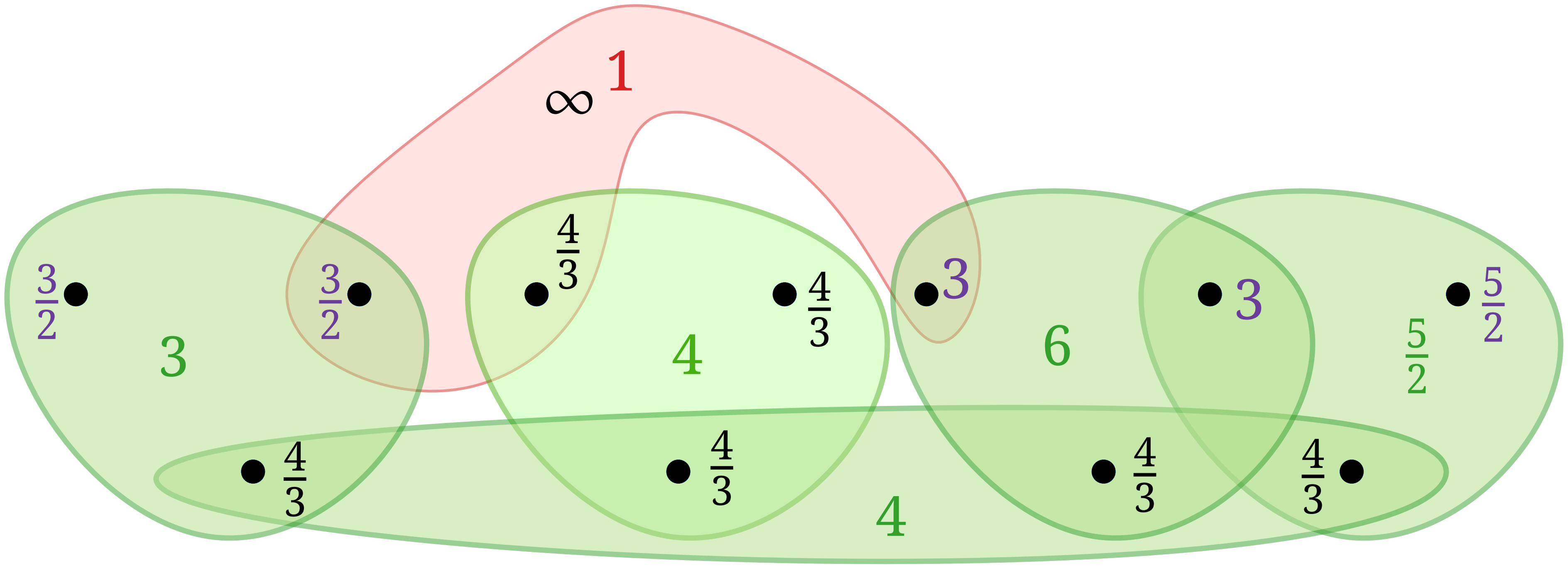
What is the real cost of picking a set?

Set with k elements and cost c has per-element cost c/k.

What happens if we “buy” a set?

Fix price of elements bought and recompute per-element cost.

total cost: $\sum_{u \in U} \mathrm{price}(u)$



Iterative "Buying" of Elements

What is the real cost of picking a set?

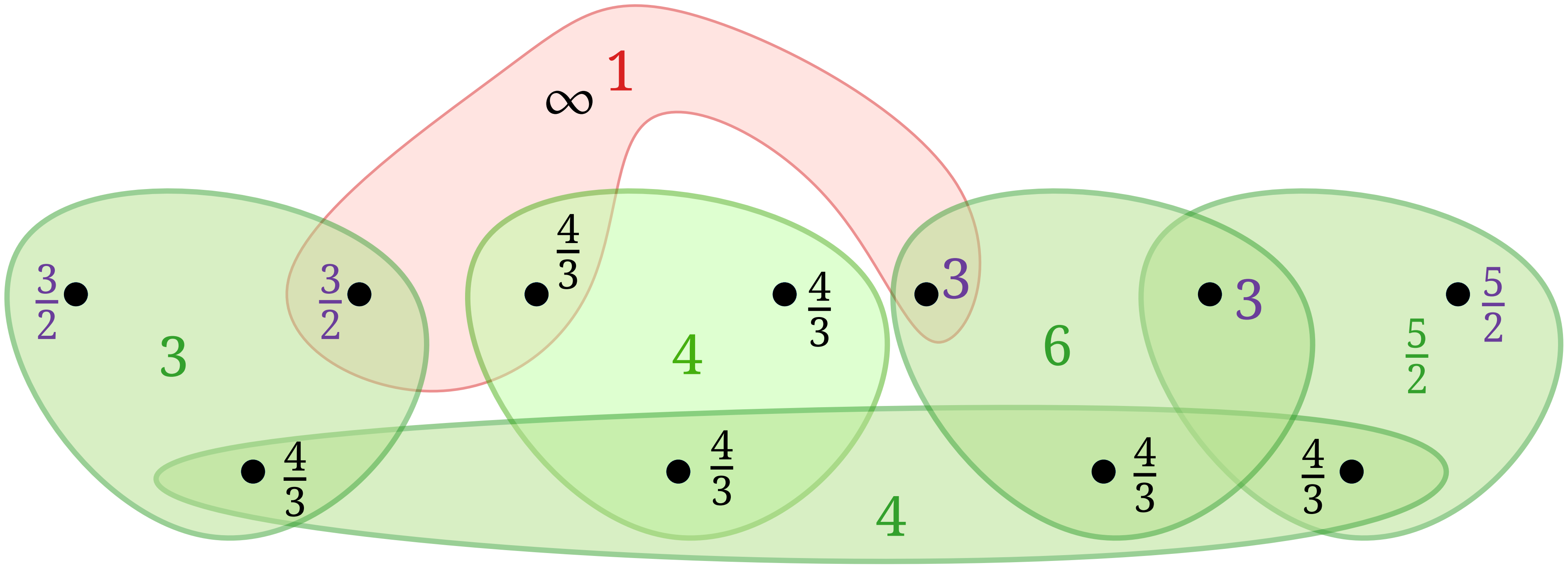
Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.

total cost: $\sum_{u \in U} \text{price}(u)$

Greedy: Always choose the set with minimum per-element cost.



Iterative “Buying” of Elements

What is the real cost of picking a set?

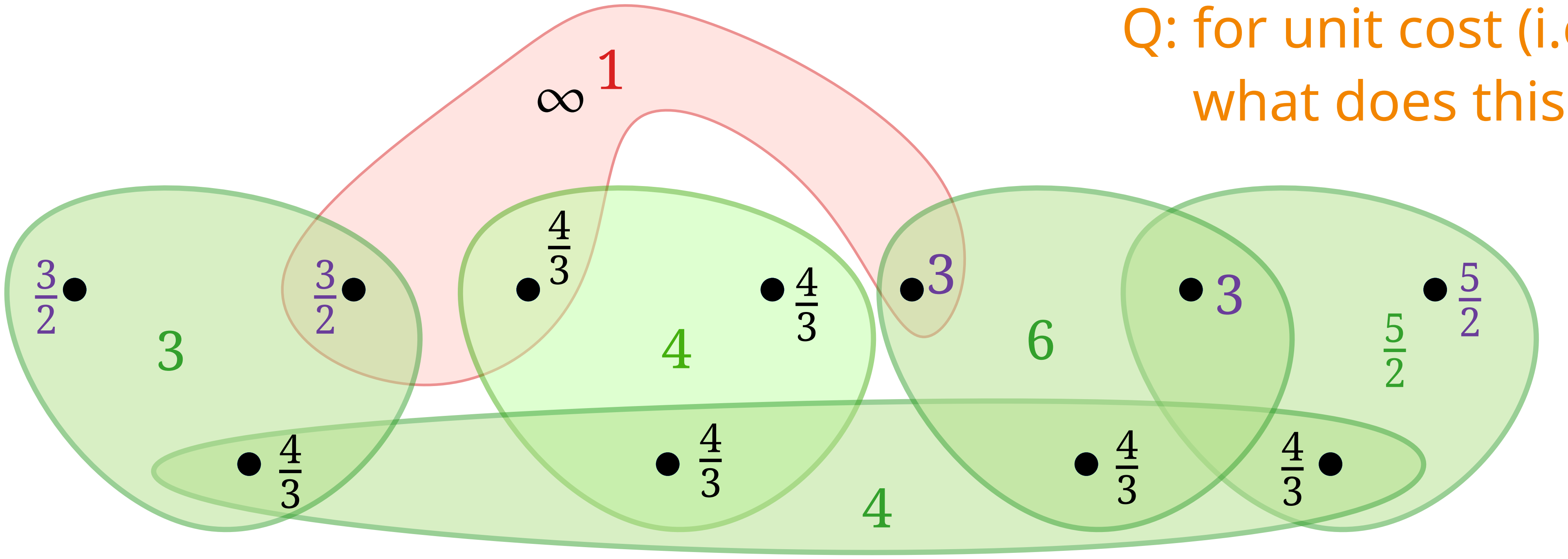
Set with k elements and cost c has per-element cost c/k.

What happens if we “buy” a set?

Fix price of elements bought and recompute per-element cost.

total cost: $\sum_{u \in U} \mathrm{price}(u)$

Greedy: Always choose the set with minimum per-element cost.



Q: for unit cost (i.e., cardinality set cover), what does this correspond to?

Iterative "Buying" of Elements

What is the real cost of picking a set?

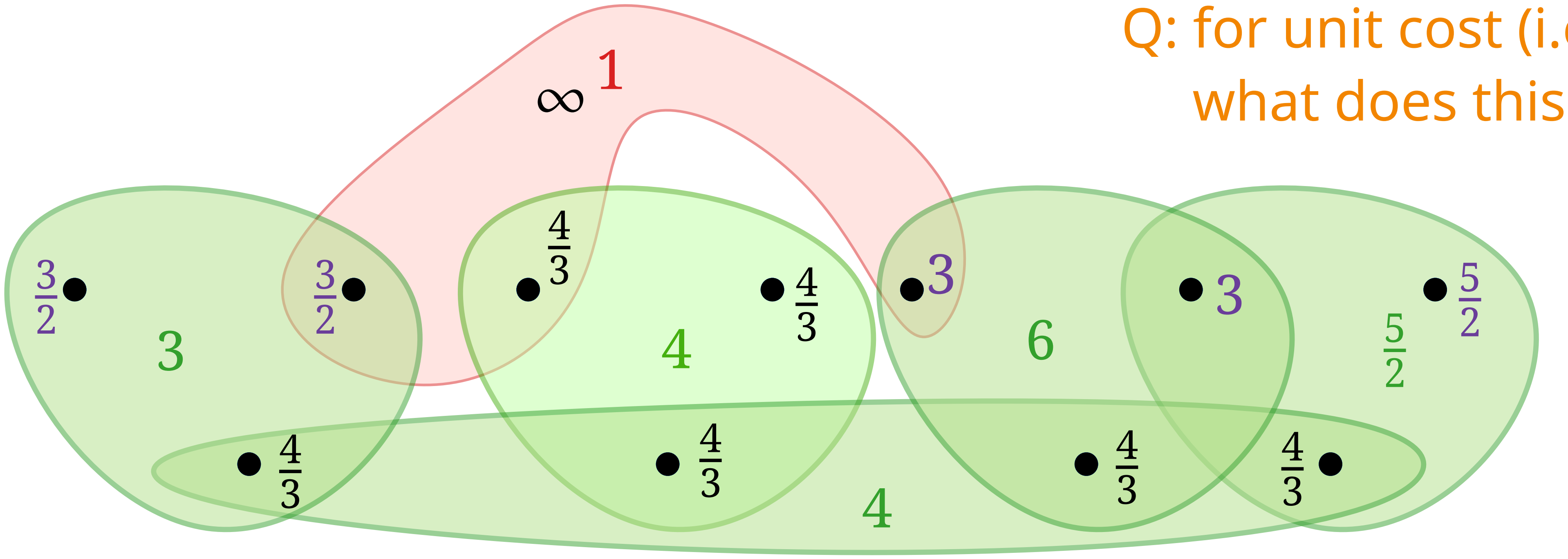
Set with k elements and cost c has per-element cost c/k.

What happens if we "buy" a set?

Fix price of elements bought and recompute per-element cost.

total cost: $\sum_{u \in U} \mathrm{price}(u)$

Greedy: Always choose the set with minimum per-element cost.



Q: for unit cost (i.e., cardinality set cover), what does this correspond to?

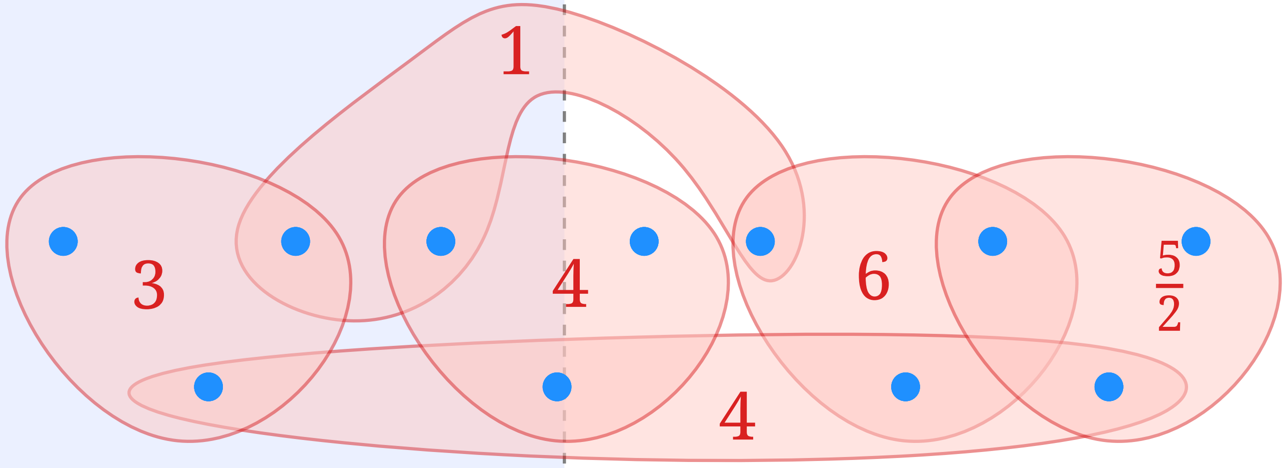
always pick the set that covers most of the remaining elements

Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$ // covered elements

$\mathcal{S}' \leftarrow \emptyset$ // picked sets



return $\mathcal{S}'$ // Cover of U

Greedy for SetCover

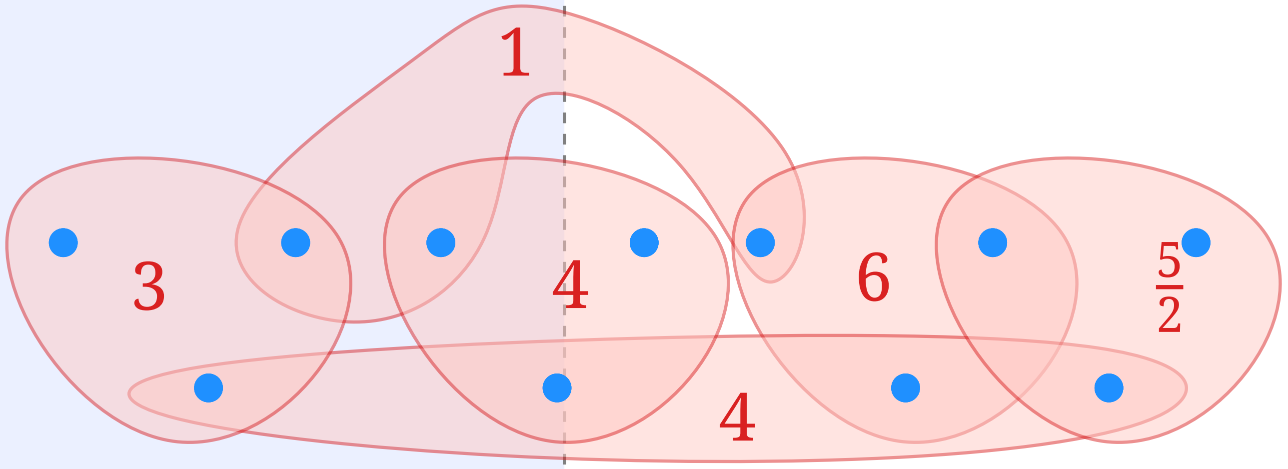
GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$ `// covered elements`

$\mathcal{S}' \leftarrow \emptyset$ `// picked sets`

while $C \neq U$ **do**

return $\mathcal{S}'$ `// Cover of` U



Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

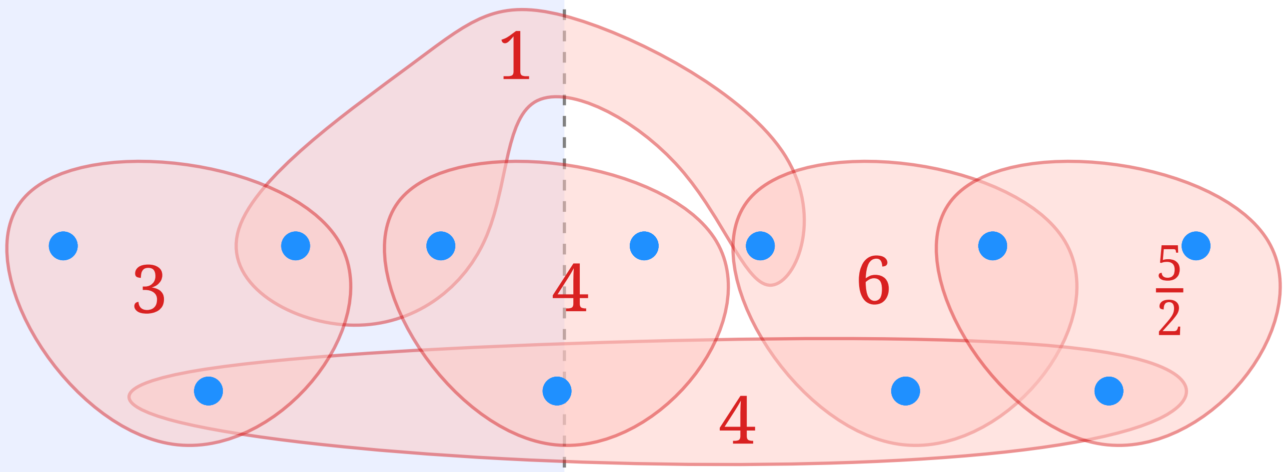
$C \leftarrow \emptyset$ `// covered elements`

$\mathcal{S}' \leftarrow \emptyset$ `// picked sets`

while $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

return $\mathcal{S}'$ `// Cover of U`



Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

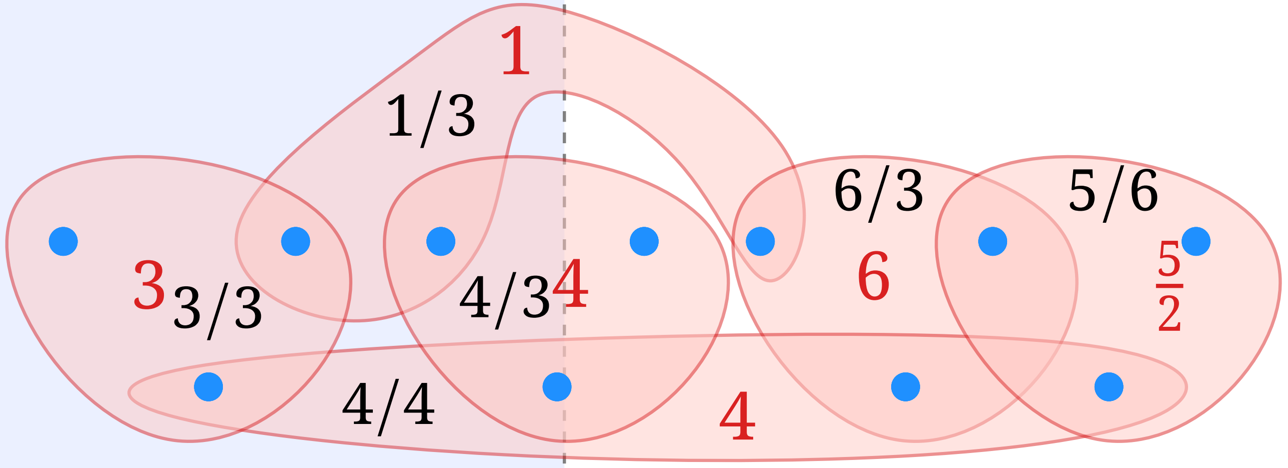
$C \leftarrow \emptyset$ // covered elements

$\mathcal{S}' \leftarrow \emptyset$ // picked sets

while $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

return $\mathcal{S}'$ // Cover of U



Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

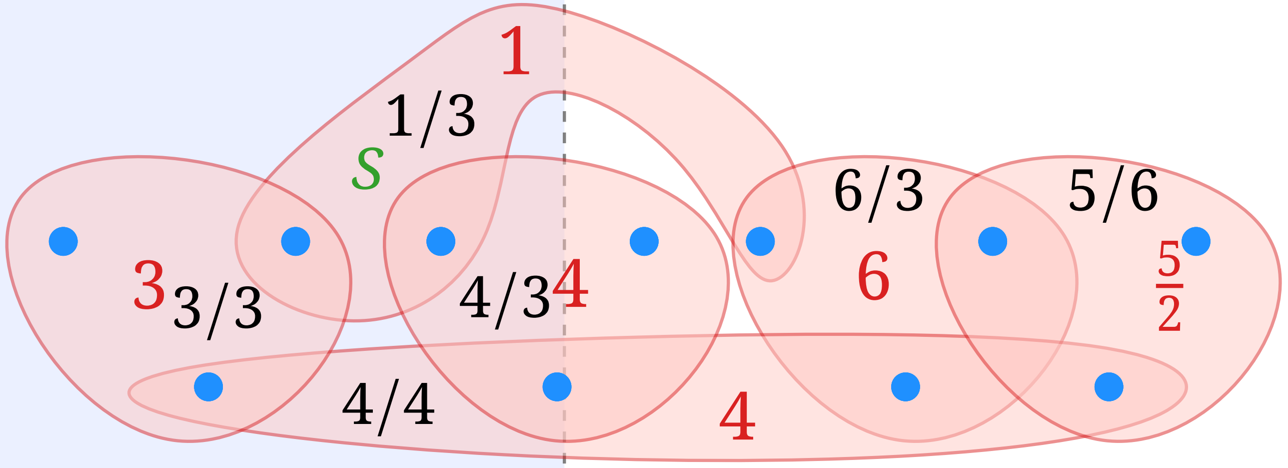
$C \leftarrow \emptyset$ `// covered elements`

$\mathcal{S}' \leftarrow \emptyset$ `// picked sets`

while $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

return $\mathcal{S}'$ `// Cover of U`



Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$ // covered elements

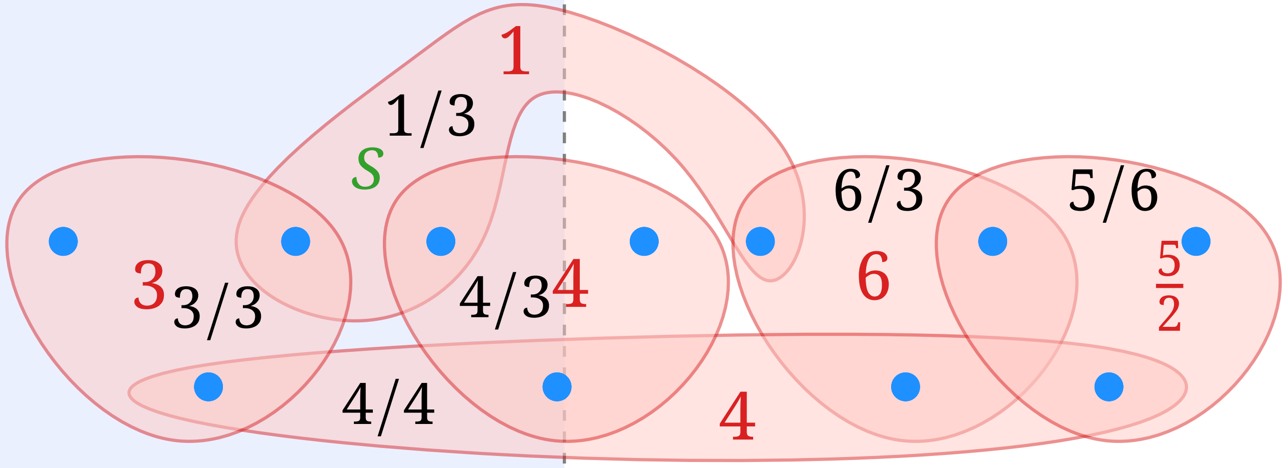
$\mathcal{S}' \leftarrow \emptyset$ // picked sets

while $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

  foreach $u \in S \setminus C$ **do**

return $\mathcal{S}'$ // Cover of U



Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$ // covered elements

$\mathcal{S}' \leftarrow \emptyset$ // picked sets

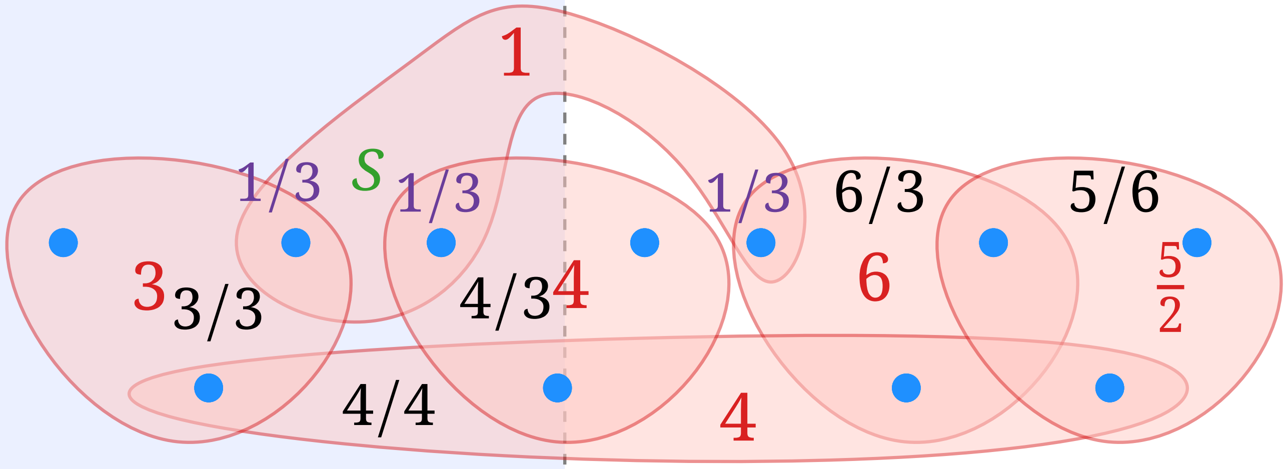
while $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

  foreach $u \in S \setminus C$ **do**

    price$(u) \leftarrow \frac{c(S)}{|S \setminus C|}$

return $\mathcal{S}'$ // Cover of U



Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$ // covered elements

$\mathcal{S}' \leftarrow \emptyset$ // picked sets

while $C \neq U$ **do**

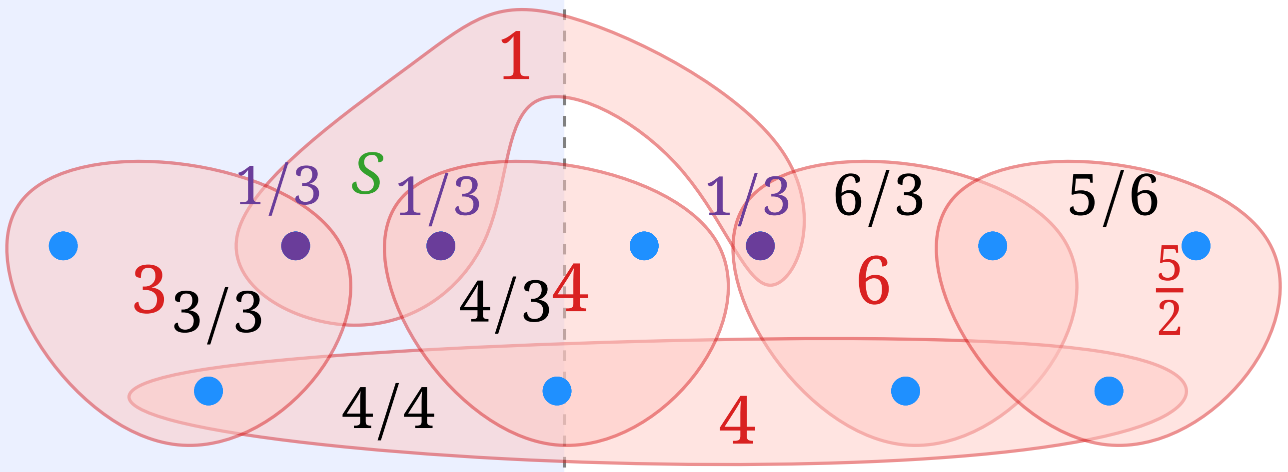
  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

  foreach $u \in S \setminus C$ **do**

    price$(u) \leftarrow \frac{c(S)}{|S \setminus C|}$

  $C \leftarrow C \cup S$

return $\mathcal{S}'$ // Cover of U



Greedy for SetCover

GreedySetCover($U, \mathcal{S}, c$)

$C \leftarrow \emptyset$ // covered elements

$\mathcal{S}' \leftarrow \emptyset$ // picked sets

while $C \neq U$ **do**

  $S \leftarrow$ set in $\mathcal{S}$ that minimizes $\frac{c(S)}{|S \setminus C|}$

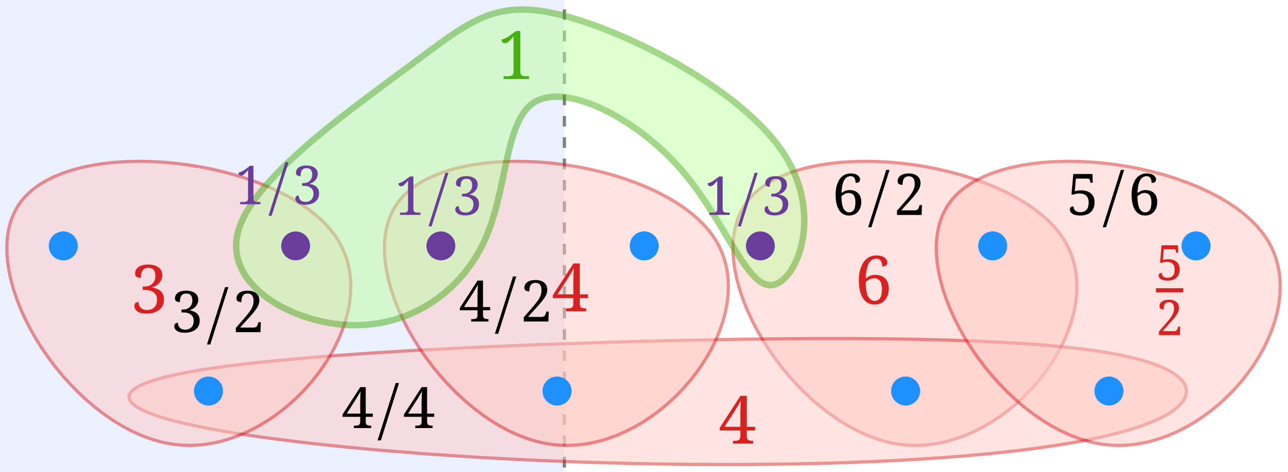
  foreach $u \in S \setminus C$ **do**

    price$(u) \leftarrow \frac{c(S)}{|S \setminus C|}$

  $C \leftarrow C \cup S$

  $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{S\}$

return $\mathcal{S}'$ // Cover of U



How does the algorithm continue?

Greedy SETCOVER: analysis

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq$

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

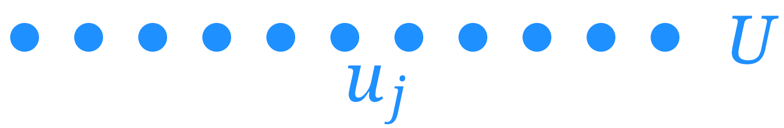
Proof.

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

Proof. Iteration at which alg. buys $u_j \Rightarrow$



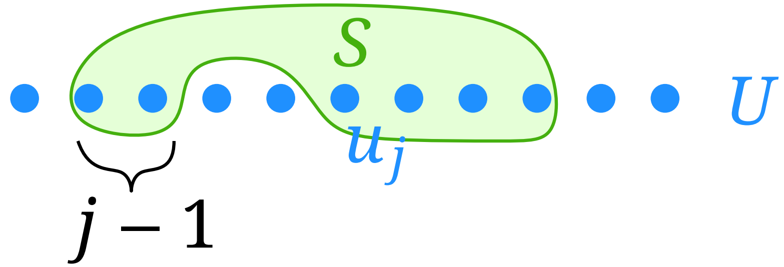
Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Proof. Iteration at which alg. buys $u_j \Rightarrow$

- $\leq j - 1$ elements of S may already be bought



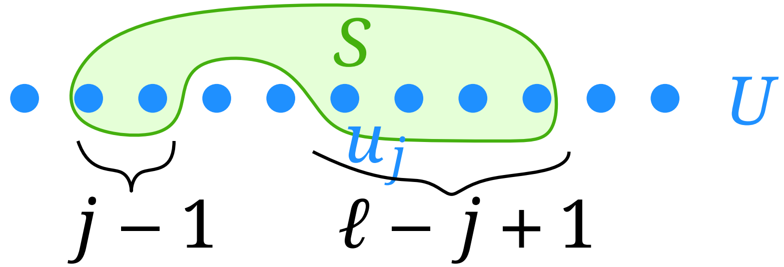
Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Proof. Iteration at which alg. buys $u_j \Rightarrow$

- $\leq j - 1$ elements of S may already be bought
- $\geq \ell - j + 1$ elements of S not yet bought



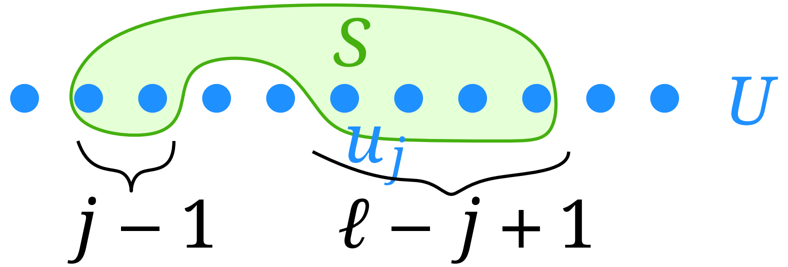
Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

Proof. Iteration at which alg. buys $u_j \Rightarrow$

- $\le j - 1$ elements of S may already be bought
- $\ge \ell - j + 1$ elements of S not yet bought
- per-element cost for S:



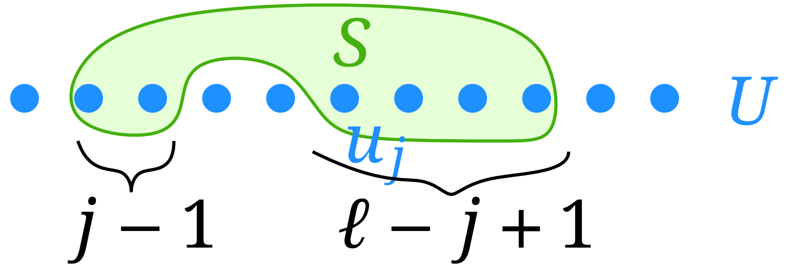
Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Proof. Iteration at which alg. buys $u_j \Rightarrow$

- $\leq j - 1$ elements of S may already be bought
- $\geq \ell - j + 1$ elements of S not yet bought
- per-element cost for S: $\leq c(S)/(\ell - j + 1)$



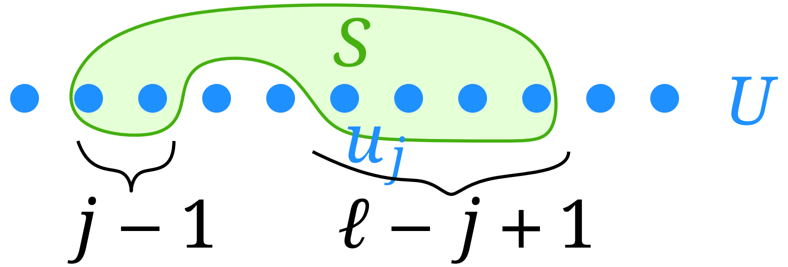
Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Proof. Iteration at which alg. buys $u_j \Rightarrow$

- $\leq j - 1$ elements of S may already be bought
- $\geq \ell - j + 1$ elements of S not yet bought
- per-element cost for S: $\leq c(S)/(\ell - j + 1)$
- price by alg. no larger due to greedy choice



Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \longrightarrow 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i)$

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Proof.

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SETCOVER, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \longrightarrow 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Proof. Let $\{S_1, \ldots, S_m\}$ be opt. sol.

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Proof. Let $\{S_1, \ldots, S_m\}$ be opt. sol. $\mathrm{OPT} = \sum_{i=1}^{m} c(S_i)$

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Proof. Let $\{S_1, \ldots, S_m\}$ be opt. sol. $\mathrm{OPT} = \sum_{i=1}^{m} c(S_i)$

Cost of solution returned by GreedySetCover:

$\mathrm{price}(U) =$

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Proof. Let $\{S_1, \ldots, S_m\}$ be opt. sol. $\mathrm{OPT} = \sum_{i=1}^{m} c(S_i)$

Cost of solution returned by GreedySetCover:

$\mathrm{price}(U) = \sum_{u \in U} \mathrm{price}(u) \leq$

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SETCOVER, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Proof. Let $\{S_1, \ldots, S_m\}$ be opt. sol. $\quad \mathrm{OPT} = \sum_{i=1}^{m} c(S_i)$

Cost of solution returned by GreedySetCover:

$$\mathrm{price}(U) = \sum_{u \in U} \mathrm{price}(u) \leq \sum_{i=1}^{m} \mathrm{price}(S_i)$$

$$\leq$$

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\text{price}(u_j) \leq c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\text{price}(S) := \sum_{i=1}^{\ell} \text{price}(u_i) \leq c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Proof. Let $\{S_1, \ldots, S_m\}$ be opt. sol. $\quad \text{OPT} = \sum_{i=1}^{m} c(S_i)$

Cost of solution returned by GreedySetCover:

$$\text{price}(U) = \sum_{u \in U} \text{price}(u) \leq \sum_{i=1}^{m} \text{price}(S_i)$$
$$\leq \sum_{i=1}^{m} c(S_i) \cdot \mathcal{H}_k =$$

Analysis

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and $\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \to 0.5 + \ln k$.

Lemma. Let $S \in \mathcal{S}$, and let $u_1, \ldots, u_\ell$ be the elements of S in the order in which they are covered ("bought") by GreedySetCover. Then $\mathrm{price}(u_j) \le c(S)/(\ell - j + 1)$.

Corollary. For $S \in \mathcal{S}$, $\mathrm{price}(S) := \sum_{i=1}^{\ell} \mathrm{price}(u_i) \le c(S) \cdot \mathcal{H}_\ell$, where $\ell = |S|$.

Proof. Let $\{S_1, \ldots, S_m\}$ be opt. sol. $\mathrm{OPT} = \sum_{i=1}^{m} c(S_i)$

Cost of solution returned by GreedySetCover:

$$\mathrm{price}(U) = \sum_{u \in U} \mathrm{price}(u) \le \sum_{i=1}^{m} \mathrm{price}(S_i)$$

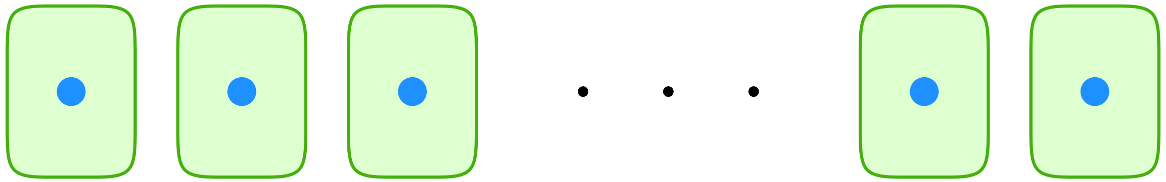
$$\le \sum_{i=1}^{m} c(S_i) \cdot \mathcal{H}_k = \mathrm{OPT} \cdot \mathcal{H}_k \qquad \square$$

Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.

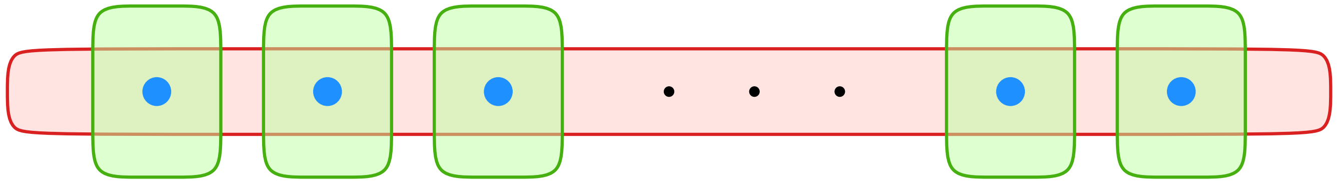
Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



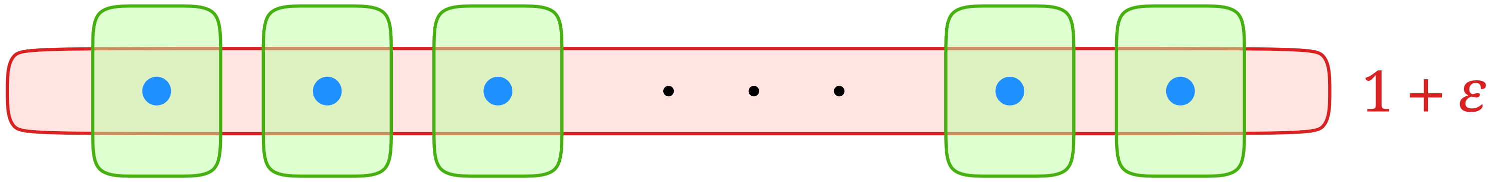
Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



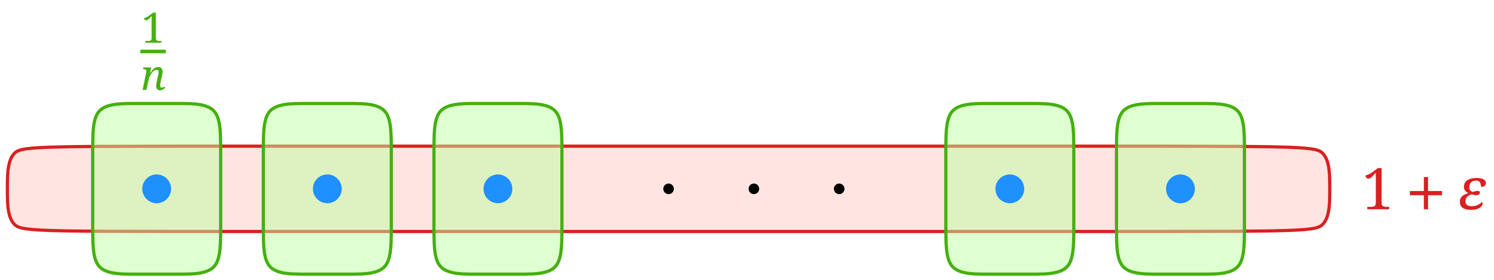
Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



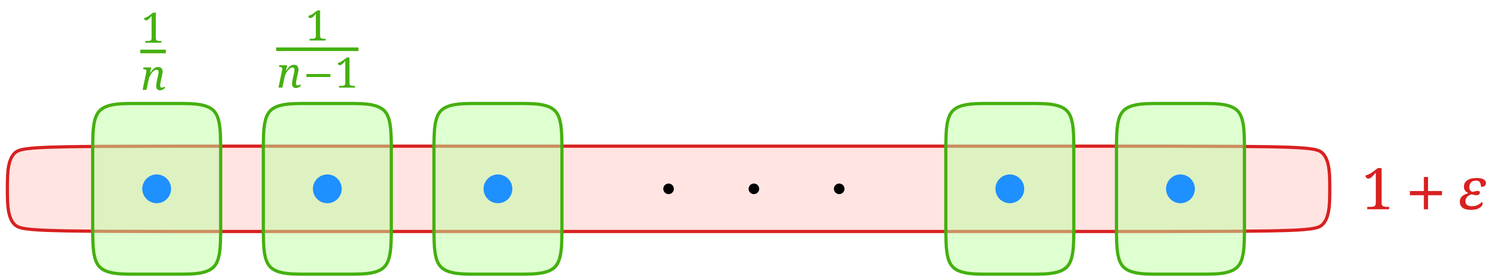
Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



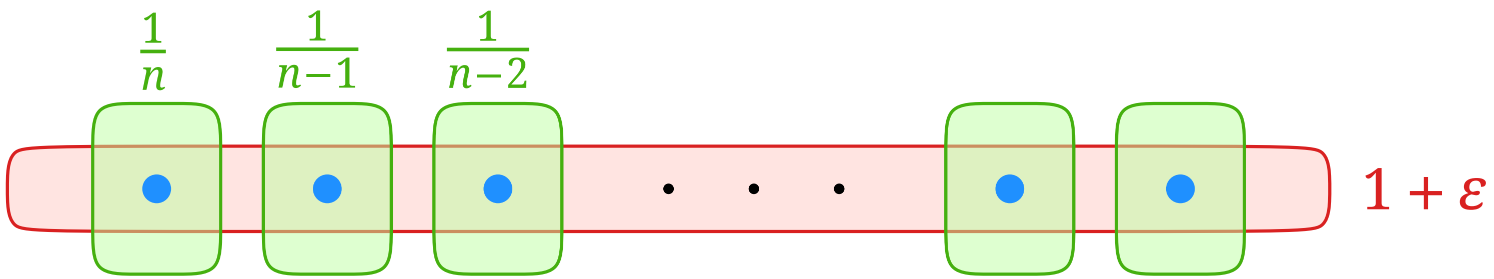
Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



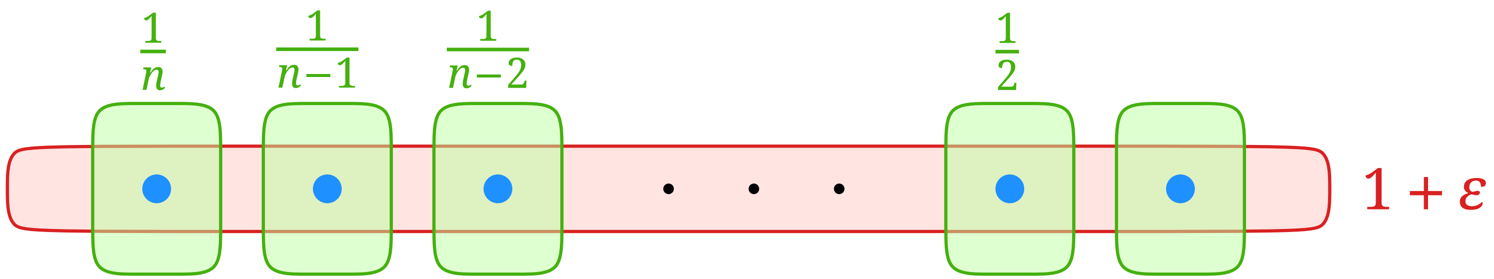
Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \le 1 + \ln k = O(\log n)$.



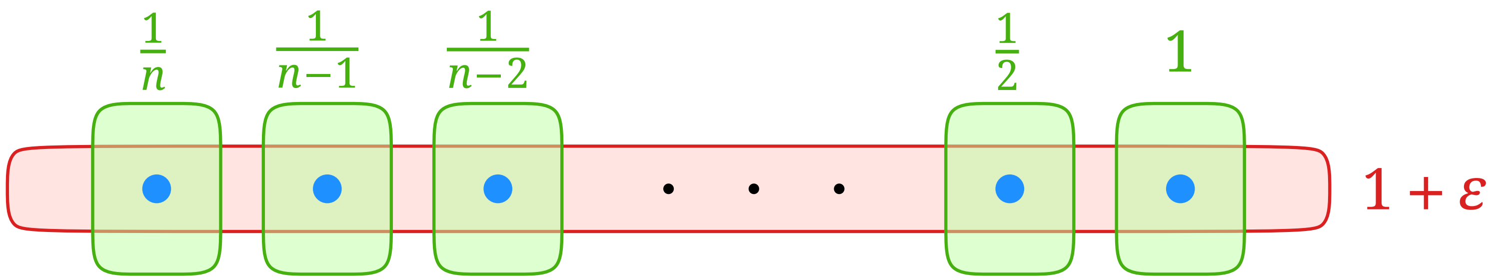
Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



Analysis tight?

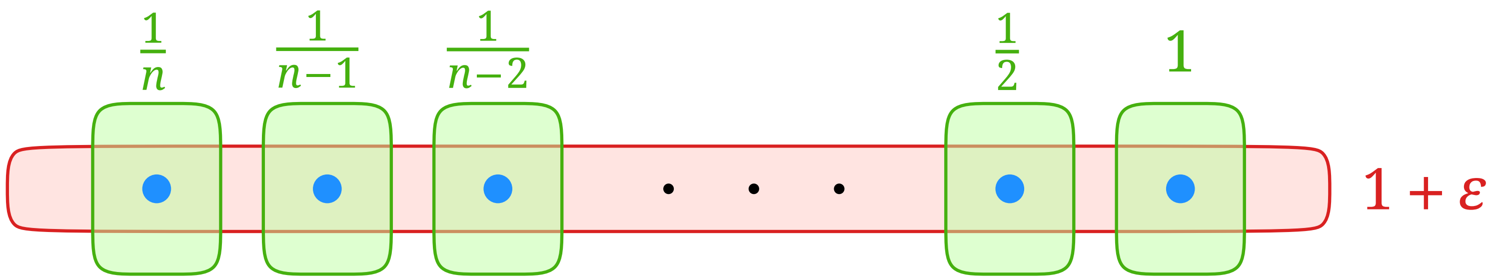
Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



Q: which sets does the algorithm choose?
what is an optimal cover?

Analysis tight?

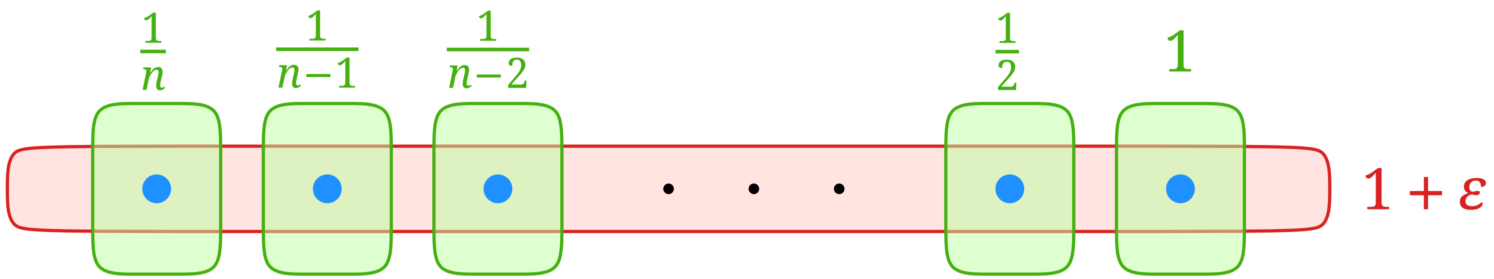
Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



$\text{price}(U) = \mathcal{H}_n$ $\quad$ $\text{OPT} = 1 + \varepsilon$

Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



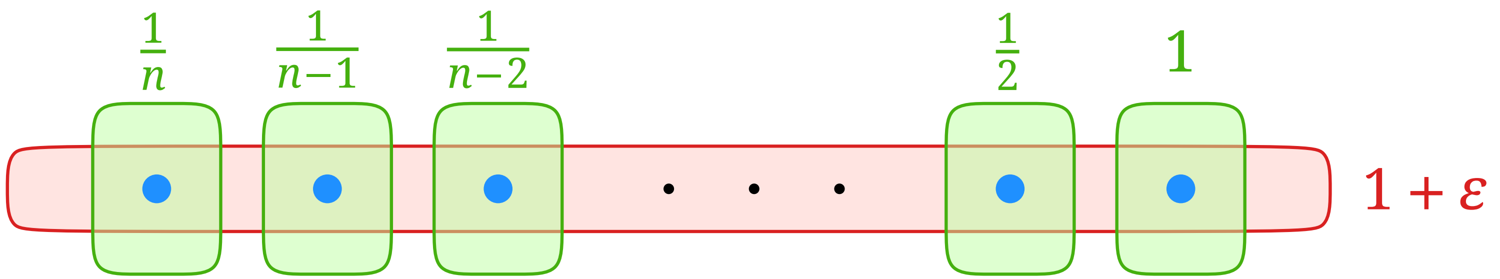
$\mathrm{price}(U) = \mathcal{H}_n$

$\mathrm{OPT} = 1 + \varepsilon$

Can we do better?

Analysis tight?

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k = O(\log n)$.



$\text{price}(U) = \mathcal{H}_n$ $\qquad$ $\text{OPT} = 1 + \varepsilon$

Can we do better?

No – SetCover cannot be approximated within factor $(1 - o(1)) \cdot \ln n$ (unless P = NP). [Feige, JACM 1998]

VERTEXCOVER as SETCOVER

Vertex cover is a special case of set cover. Q: How?

VertexCover as SetCover

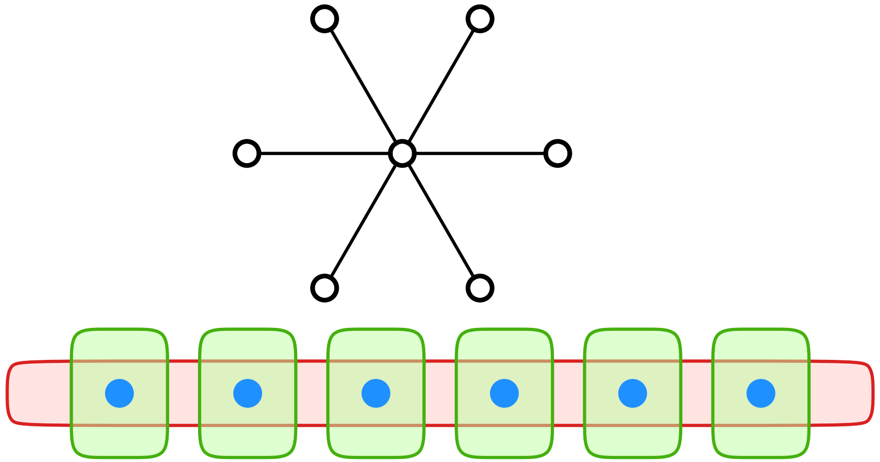
Vertex cover is a special case of set cover.

Given graph $G = (V, E)$
let $U = E$ and $S = \{S_1, \ldots, S_n\}$ where $S_i = \{e \in U \mid v_i \in e\}$

VertexCover as SetCover

Vertex cover is a special case of set cover.

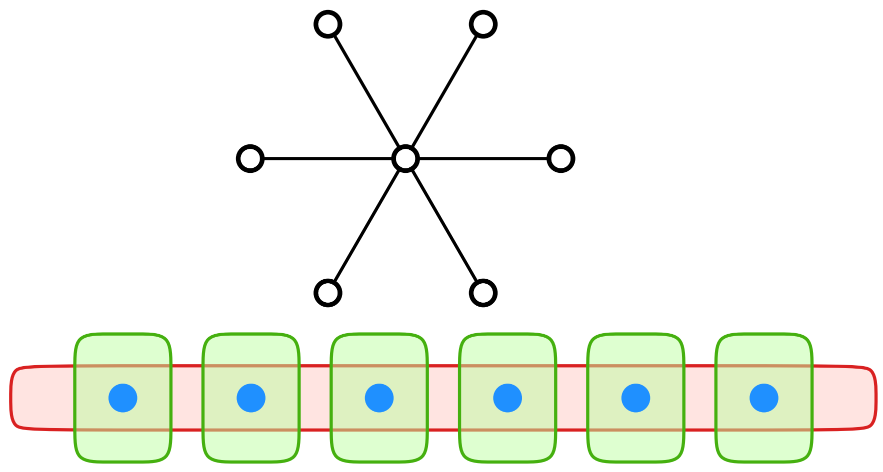
Given graph $G = (V, E)$
let $U = E$ and $S = \{S_1, \ldots, S_n\}$ where $S_i = \{e \in U \mid v_i \in e\}$



VertexCover as SetCover

Vertex cover is a special case of set cover.

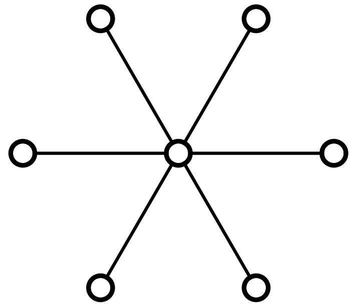
Given graph $G = (V, E)$
let $U = E$ and $S = \{S_1, \ldots, S_n\}$ where $S_i = \{e \in U \mid v_i \in e\}$



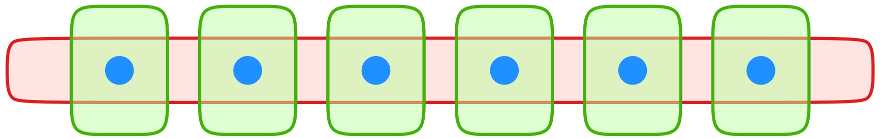
Q: How does the greedy algorithm for set cover work for vertex cover?

VERTEXCOVER as SETCOVER

Vertex cover is a special case of set cover.

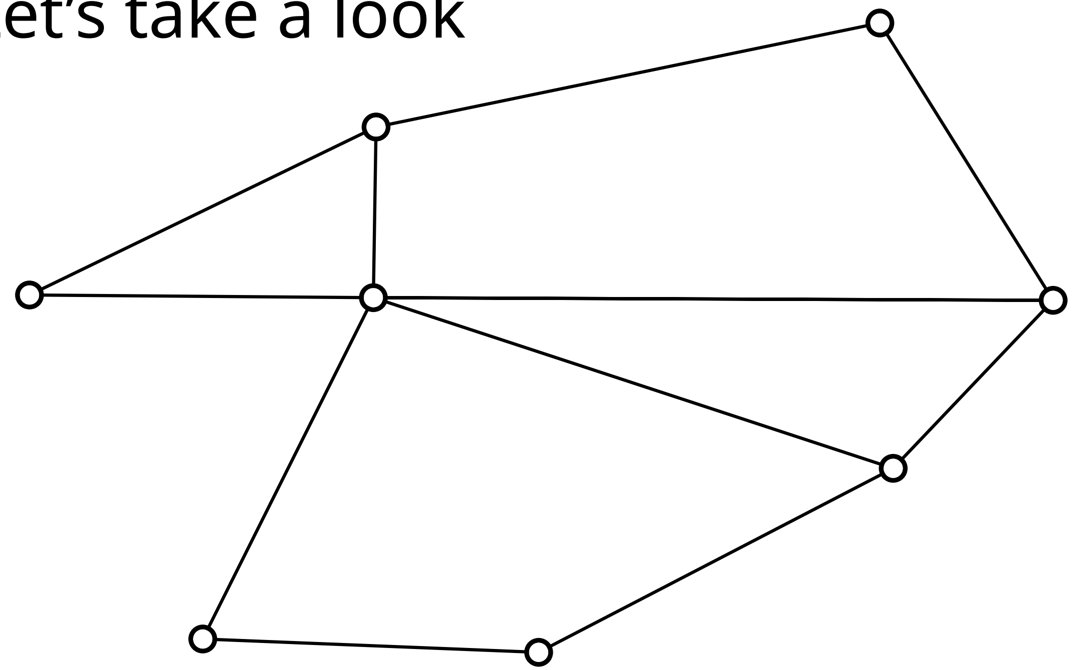


Given graph $G = (V, E)$
let $U = E$ and $S = \{S_1, \ldots, S_n\}$ where $S_i = \{e \in U \mid v_i \in e\}$



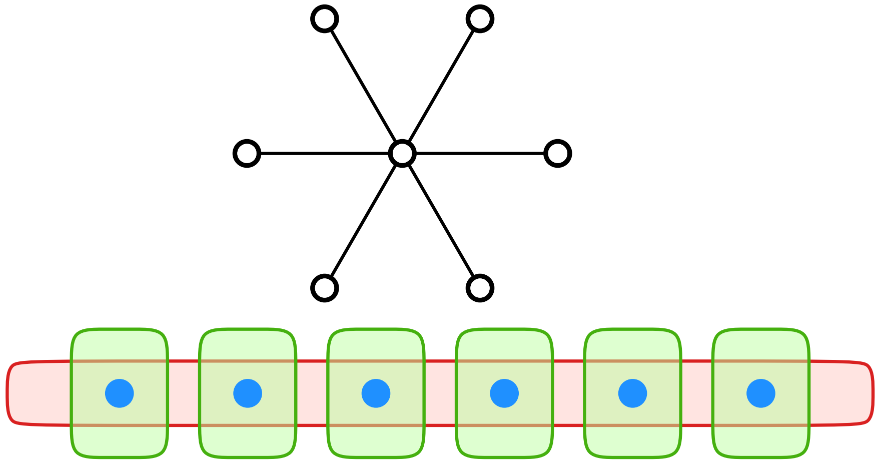
Q: How does the greedy algorithm for set cover work for vertex cover?

Let's take a look



VertexCover as SetCover

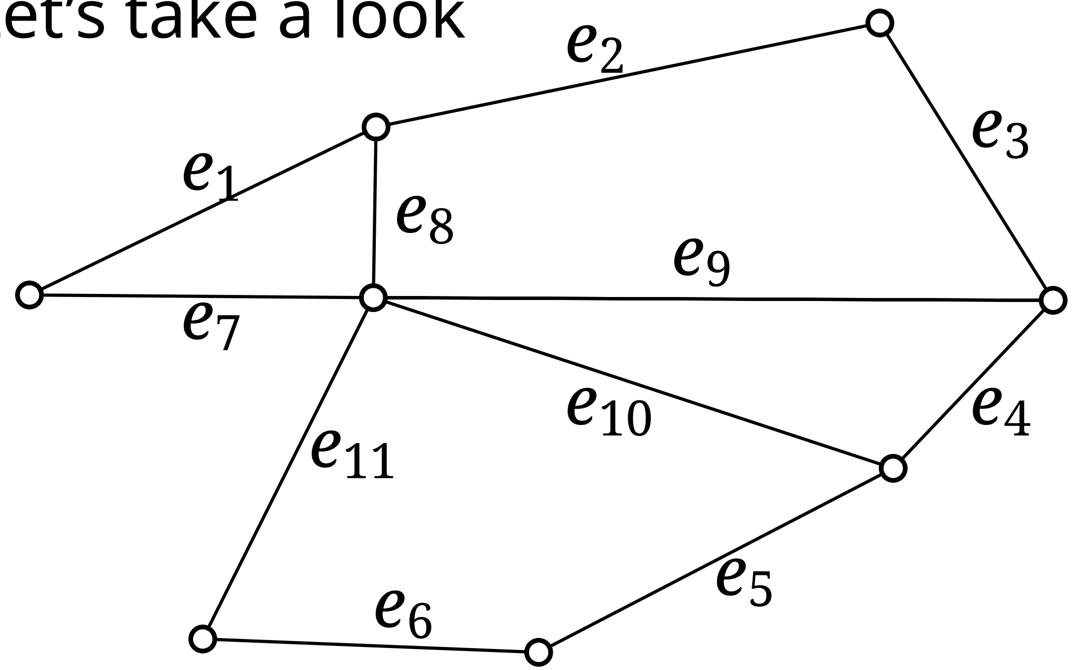
Vertex cover is a special case of set cover.



Given graph $G = (V, E)$
let $U = E$ and $S = \{S_1, \ldots, S_n\}$ where $S_i = \{e \in U \mid v_i \in e\}$

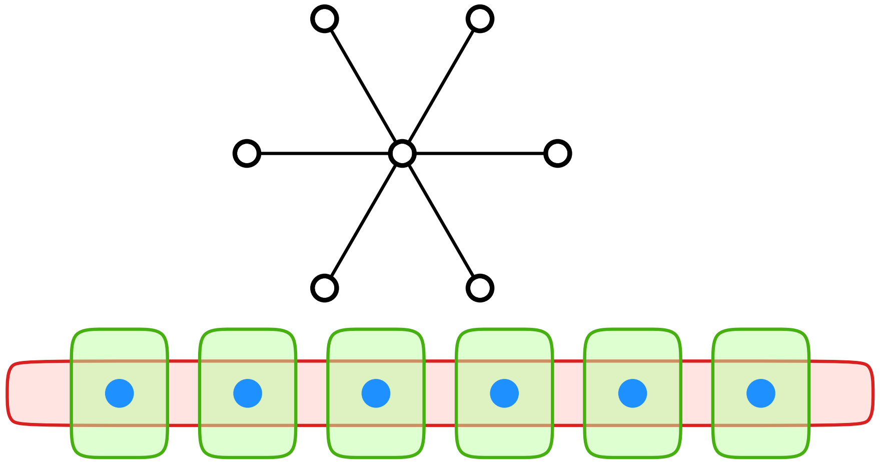
Q: How does the greedy algorithm for set cover work for vertex cover?

Let's take a look



VertexCover as SetCover

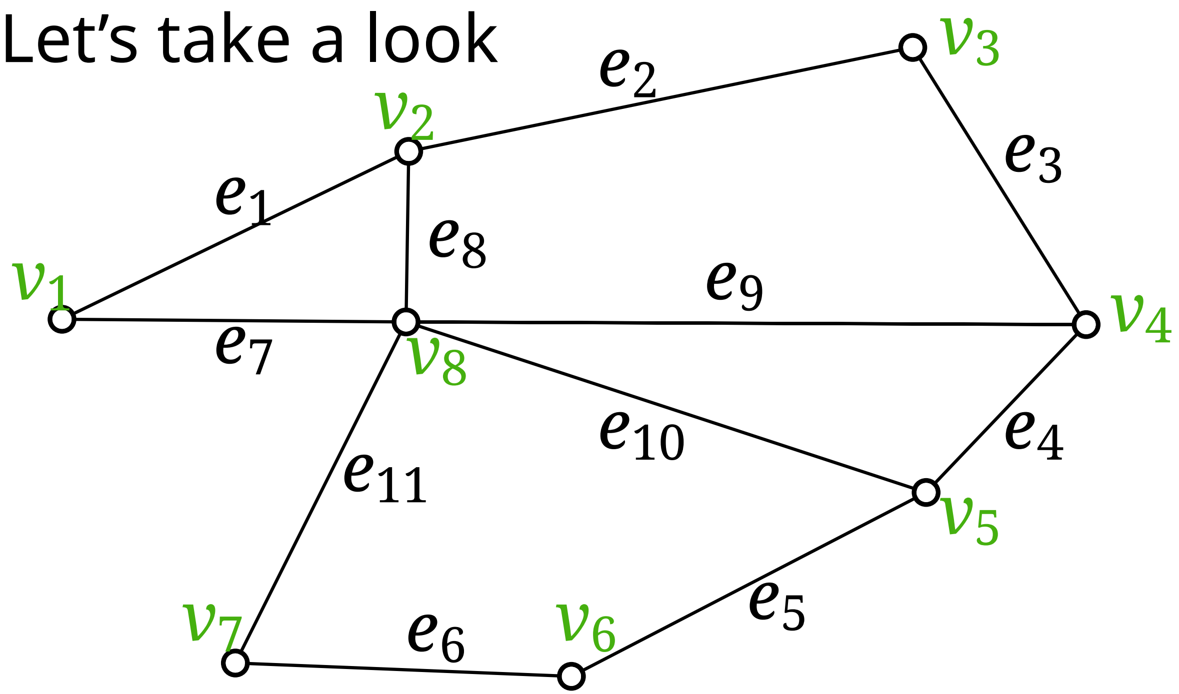
Vertex cover is a special case of set cover.

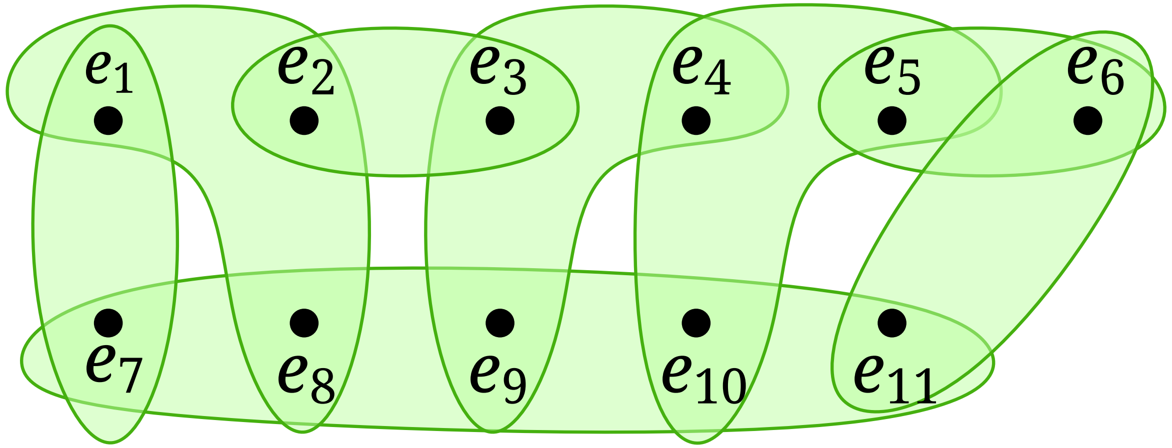


Given graph $G = (V, E)$
let $U = E$ and $S = \{S_1, \ldots, S_n\}$ where $S_i = \{e \in U \mid v_i \in e\}$

Q: How does the greedy algorithm for set cover work for vertex cover?

Let's take a look

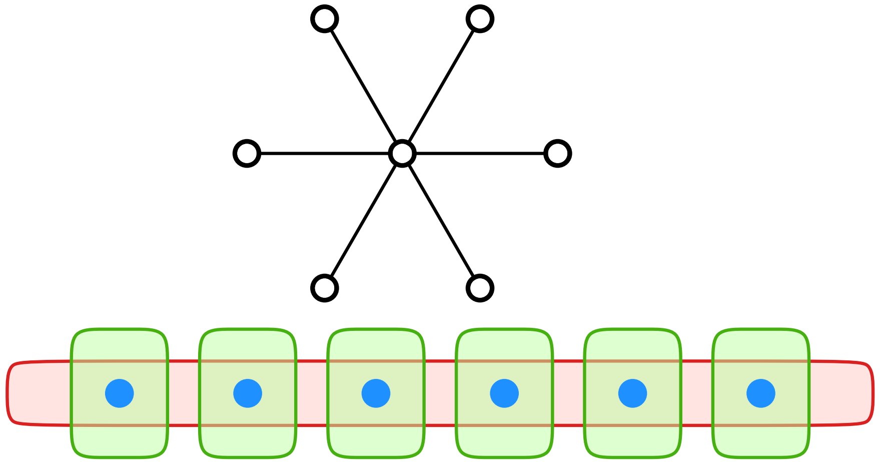




VertexCover as SetCover

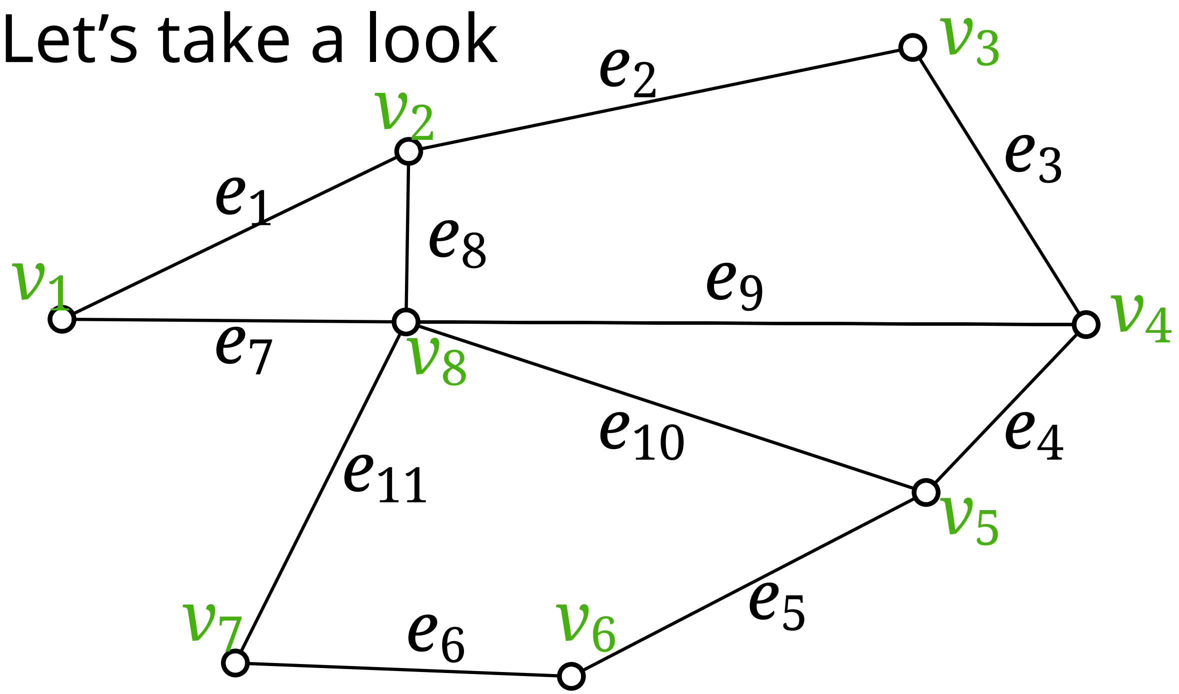
Vertex cover is a special case of set cover.

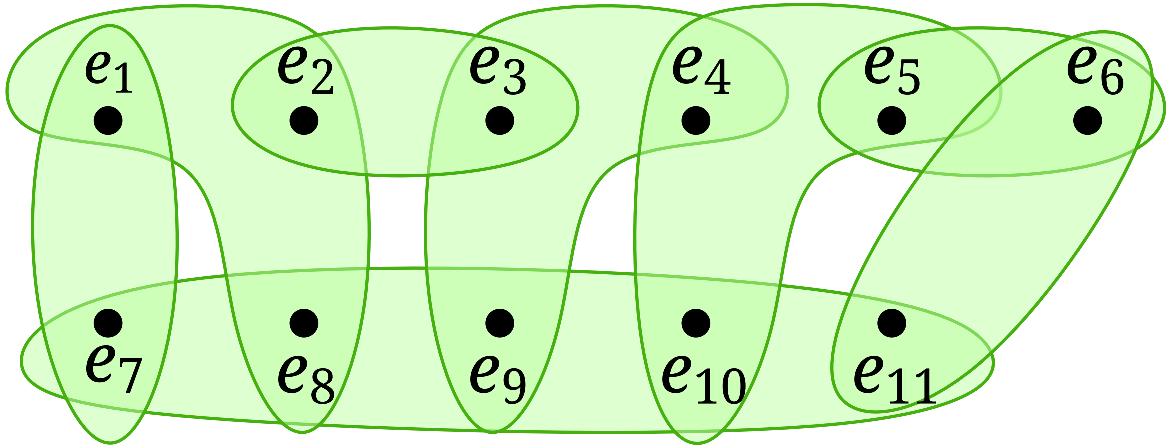
Given graph $G = (V, E)$
let $U = E$ and $S = \{S_1, \ldots, S_n\}$ where $S_i = \{e \in U \mid v_i \in e\}$



Q: How does the greedy algorithm for set cover work for vertex cover?

Let's take a look



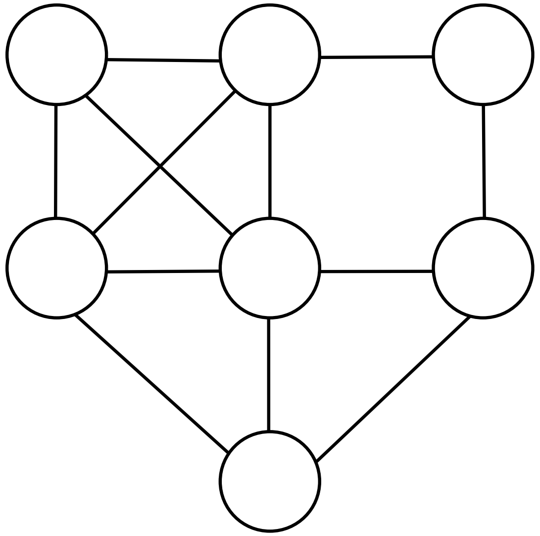


The greedy algorithm always chooses the vertex with largest remaining degree!

Layering Algorithm for Weighted VertexCover

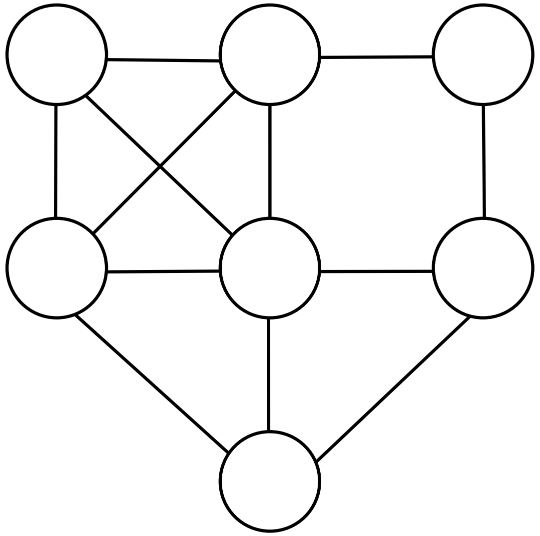
Approximation Algorithm for Weighted VERTEXCOVER

Example:



Approximation Algorithm for Weighted VERTEXCOVER

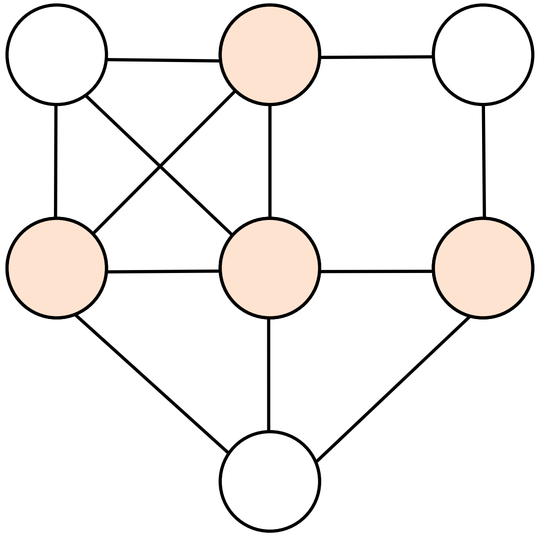
Example:



Q: What is a min cardinality vertex cover here?

Approximation Algorithm for Weighted VERTEXCOVER

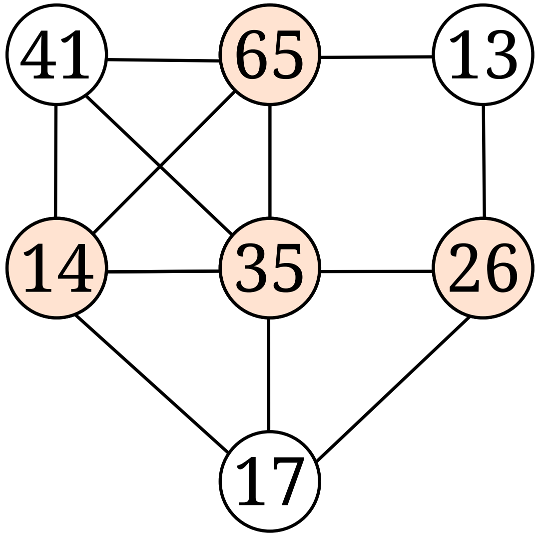
Example:



Q: What is a min cardinality vertex cover here?

Approximation Algorithm for Weighted VERTEXCOVER

Example:

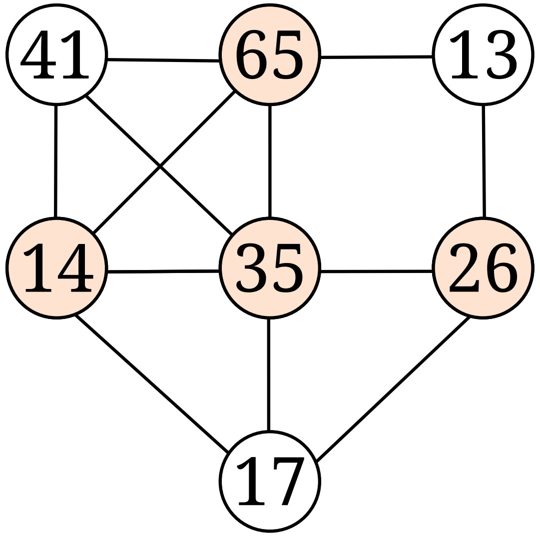


Q: What is a min cardinality vertex cover here?

Is this also a minimum weight vertex cover?

Approximation Algorithm for Weighted VERTEXCOVER

Example:



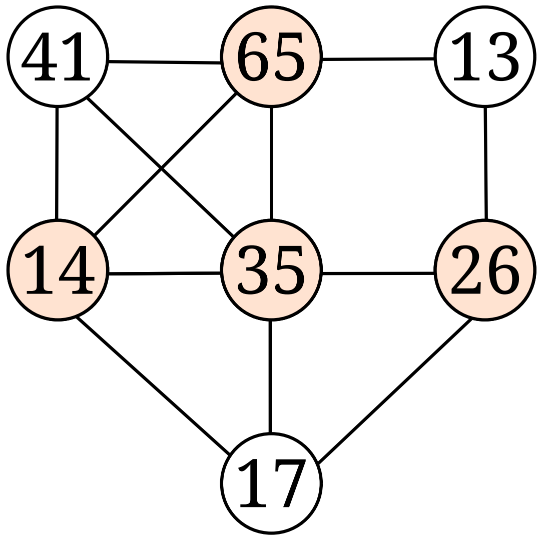
Q: What is a min cardinality vertex cover here?

Is this also a minimum weight vertex cover? No!

Does 2-approx work with weights?

Approximation Algorithm for Weighted VERTEXCOVER

Example:



Q: What is a min cardinality vertex cover here?

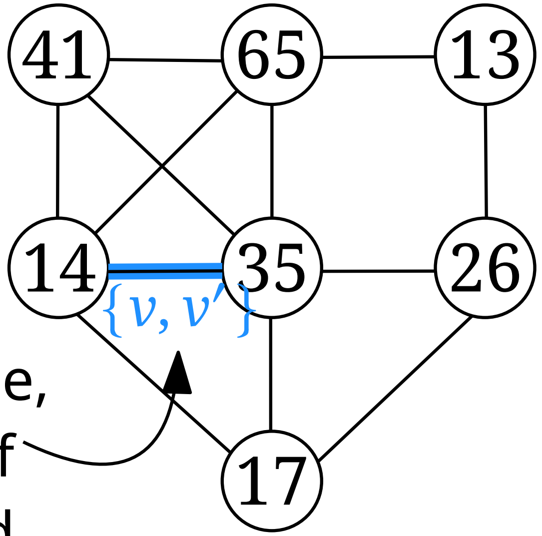
Is this also a minimum weight vertex cover? No!

Does 2-approx work with weights? No!

Now instead the approach of local ratio / layering!

Approximation Algorithm for Weighted VertexCover

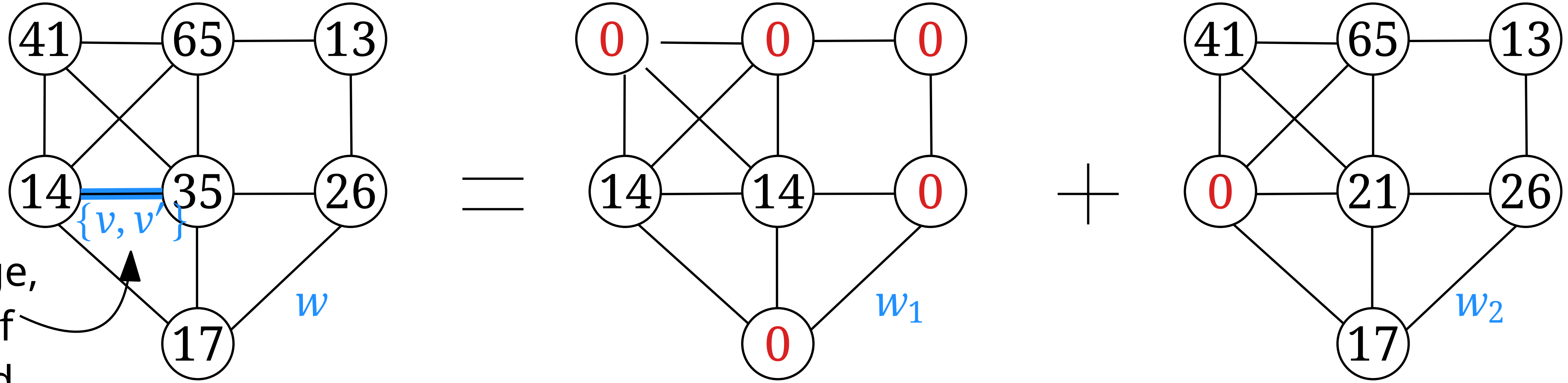
Example:



to cover this edge,
at least a price of
14 has to be paid

Approximation Algorithm for Weighted VertexCover

Example:

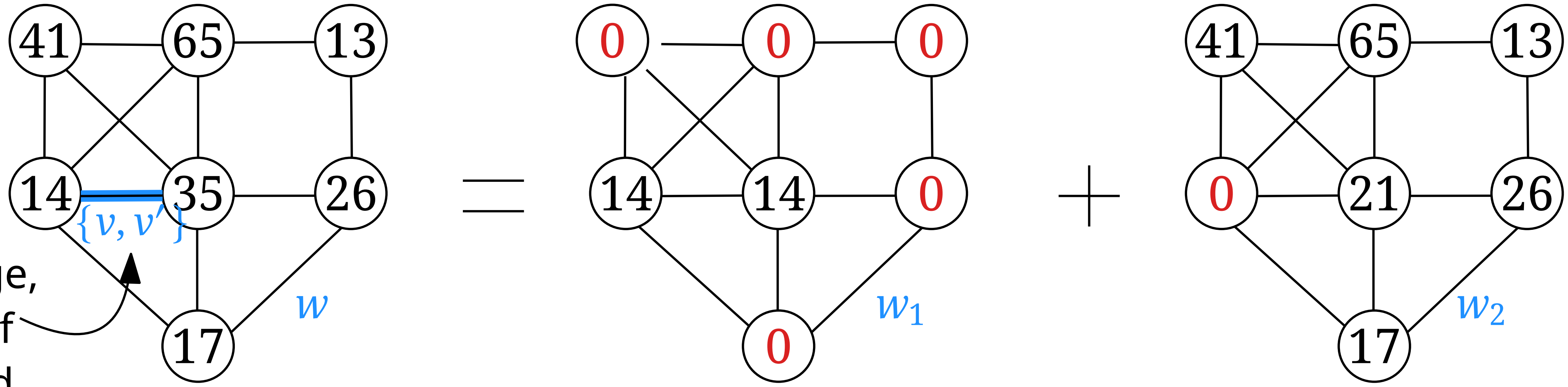


to cover this edge,
at least a price of
14 has to be paid

decompose weights: $w = w_1 + w_2$, and
consider instances I_w, I_{w_1}, and I_{w_2}

Approximation Algorithm for Weighted VertexCover

Example:



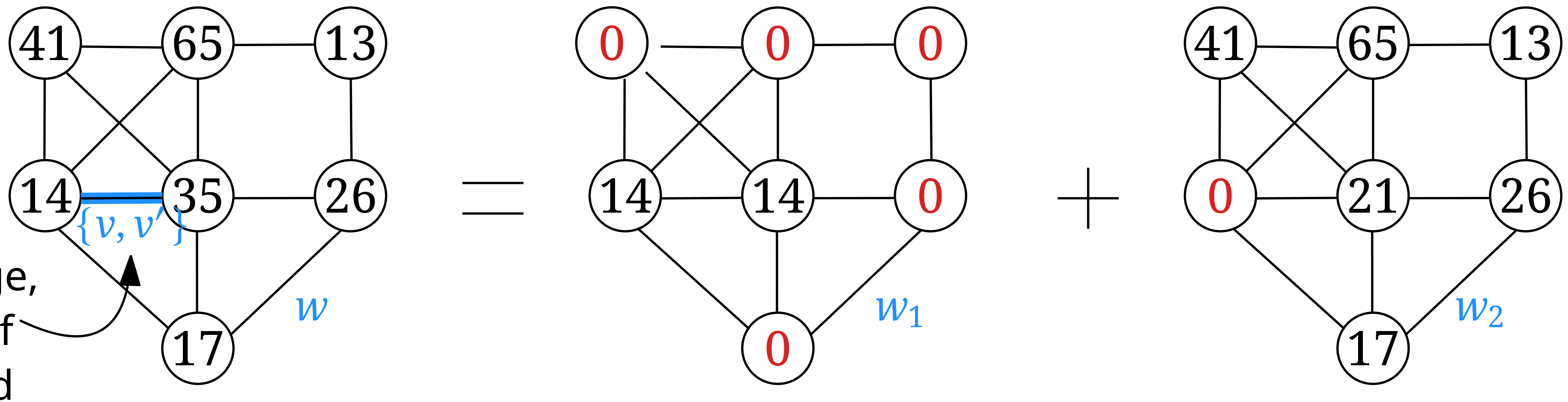
to cover this edge,
at least a price of
14 has to be paid

decompose weights: $w = w_1 + w_2$, and
consider instances I_w, I_{w_1}, and I_{w_2}

$w_1(v) = w_1(v') = \min(w(v), w(v'))$,
$w_1(u) = 0$ for $u \neq v, v'$

Approximation Algorithm for Weighted VERTEXCOVER

Example:



to cover this edge, at least a price of 14 has to be paid

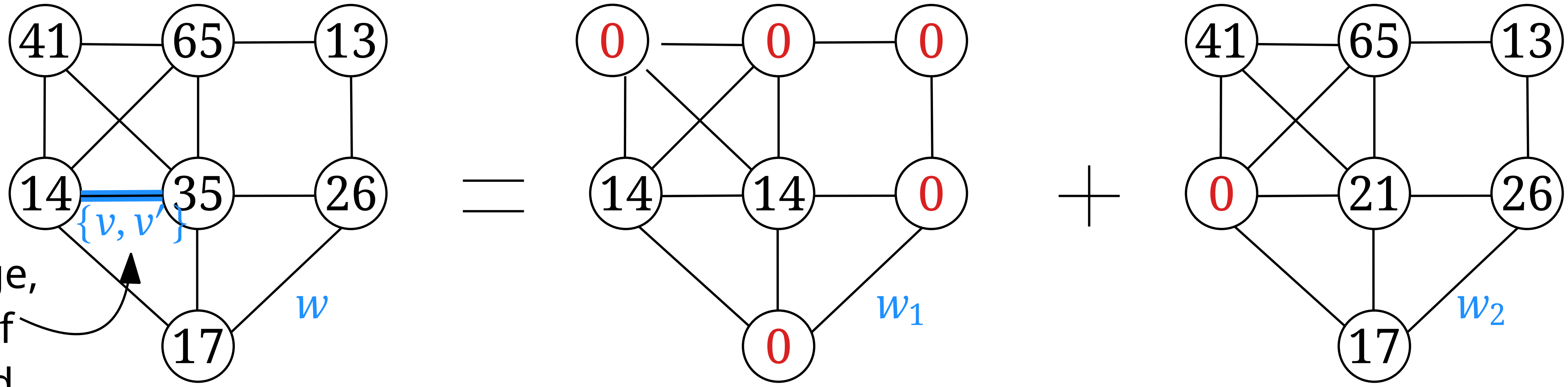
decompose weights: $w = w_1 + w_2$, and consider instances I_w, I_{w_1}, and I_{w_2}

any vertex cover for w_1 has weight 14 or 28

$w_1(v) = w_1(v') = \min(w(v), w(v'))$,
$w_1(u) = 0$ for $u \neq v, v'$

Approximation Algorithm for Weighted VERTEXCOVER

Example:



to cover this edge, at least a price of 14 has to be paid

decompose weights: $w = w_1 + w_2$, and consider instances I_w, I_{w_1}, and I_{w_2}

$w_1(v) = w_1(v') = \min(w(v), w(v'))$,
$w_1(u) = 0$ for $u \neq v, v'$

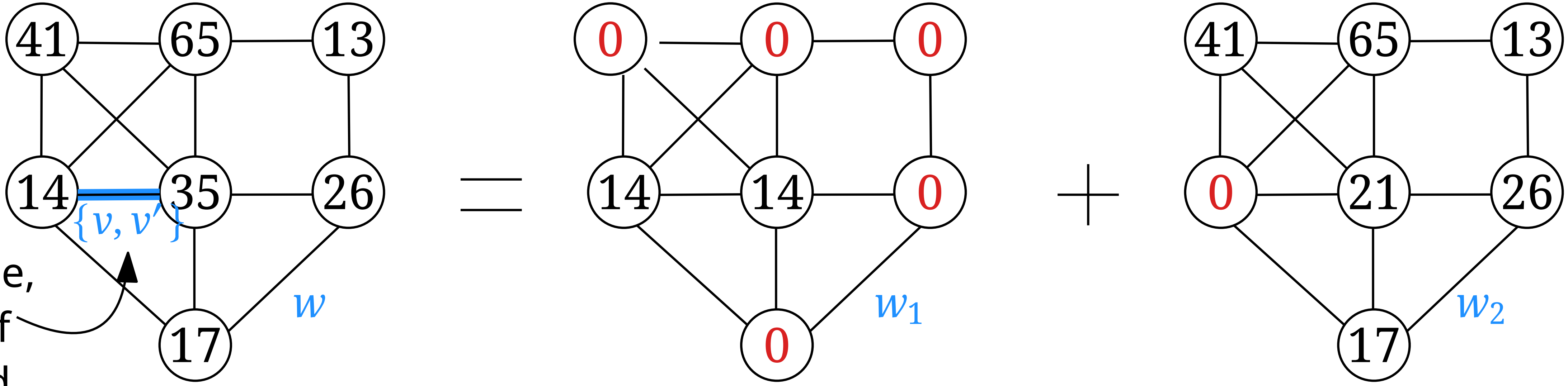
any vertex cover for w_1 has weight 14 or 28

$\Downarrow$

any vertex cover is a 2-approximation for w_1

Approximation Algorithm for Weighted VERTEXCOVER

Example:



to cover this edge, at least a price of 14 has to be paid

decompose weights: $w = w_1 + w_2$, and consider instances I_w, I_{w_1}, and I_{w_2}

$w_1(v) = w_1(v') = \min(w(v), w(v'))$,
$w_1(u) = 0$ for $u \neq v, v'$

any vertex cover for w_1 has weight 14 or 28

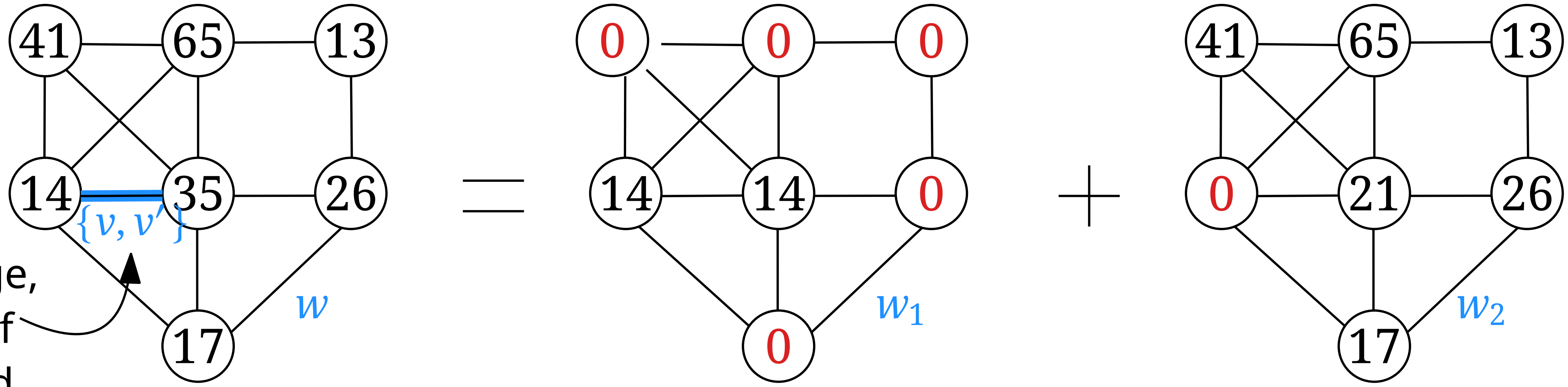
$\Downarrow$

any vertex cover is a 2-approximation for w_1

(recursively) compute a 2-approximation for w_2

Approximation Algorithm for Weighted VERTEXCOVER

Example:



to cover this edge,
at least a price of
14 has to be paid

decompose weights: $w = w_1 + w_2$, and consider instances I_w, I_{w_1}, and I_{w_2}

$w_1(v) = w_1(v') = \min(w(v), w(v'))$,
$w_1(u) = 0$ for $u \neq v, v'$

any vertex cover for w_1 has weight 14 or 28

$\Downarrow$

any vertex cover is a 2-approximation for w_1

(recursively) compute a 2-approximation for w_2

Claim: A 2-approximation for w_2 is a 2-approximation for w

Local Ratio Theorem

Let $w = w_1 + w_2$. If x is an r-approximate solution for w_1 and w_2 then x is r-approximate with respect to w as well.

Local Ratio Theorem

Let $w = w_1 + w_2$. If x is an r-approximate solution for w_1 and w_2 then x is r-approximate with respect to w as well.

Proof.

Let x^*, x_1^* and x_2^* be optimal solutions for w, w_1 and w_2, respectively

Local Ratio Theorem

Let $w = w_1 + w_2$. If x is an r-approximate solution for w_1 and w_2 then x is r-approximate with respect to w as well.

Proof.

Let x^*, x_1^* and x_2^* be optimal solutions for w, w_1 and w_2, respectively

$$w_1(x) \leq r \cdot w_1(x_1^*)$$

Local Ratio Theorem

Let $w = w_1 + w_2$. If x is an r-approximate solution for w_1 and w_2 then x is r-approximate with respect to w as well.

Proof.

Let x^*, x_1^* and x_2^* be optimal solutions for w, w_1 and w_2, respectively

$$w_1(x) \le r \cdot w_1(x_1^*) \le r \cdot w_1(x^*)$$

Local Ratio Theorem

Let $w = w_1 + w_2$. If x is an r-approximate solution for w_1 and w_2 then x is r-approximate with respect to w as well.

Proof.

Let x^*, x_1^* and x_2^* be optimal solutions for w, w_1 and w_2, respectively

$$w_1(x) \leq r \cdot w_1(x_1^*) \leq r \cdot w_1(x^*)$$

$$w_2(x) \leq r \cdot w_2(x_2^*) \leq r \cdot w_2(x^*)$$

Local Ratio Theorem

Let $w = w_1 + w_2$. If x is an r-approximate solution for w_1 and w_2 then x is r-approximate with respect to w as well.

Proof.

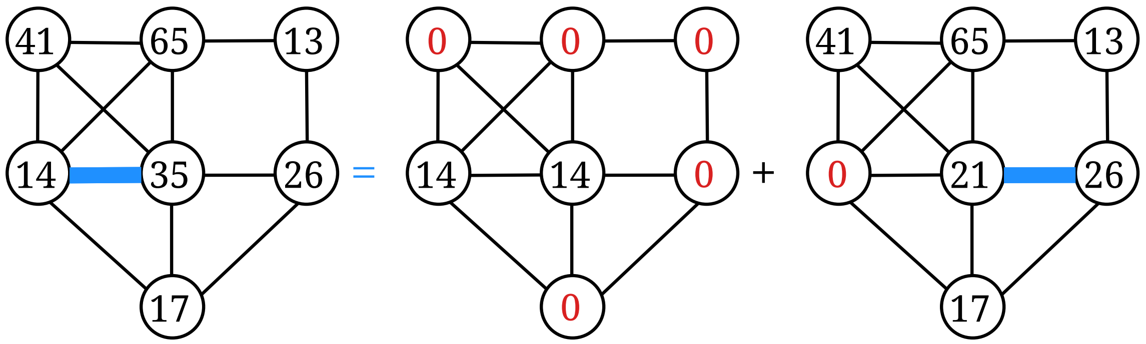
Let x^*, x_1^* and x_2^* be optimal solutions for w, w_1 and w_2, respectively

$$w_1(x) \leq r \cdot w_1(x_1^*) \leq r \cdot w_1(x^*)$$

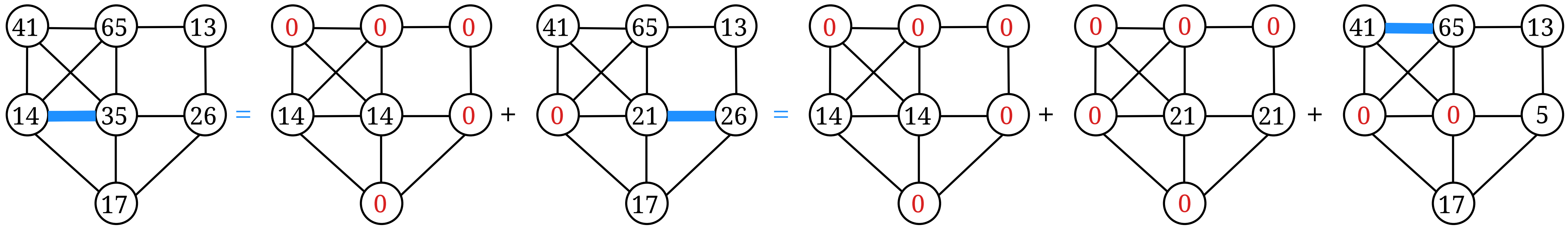
$$w_2(x) \leq r \cdot w_2(x_2^*) \leq r \cdot w_2(x^*)$$

$$w(x) = w_1(x) + w_2(x) \leq r \cdot (w_1(x^*) + w_2(x^*)) = r \cdot w(x^*)$$

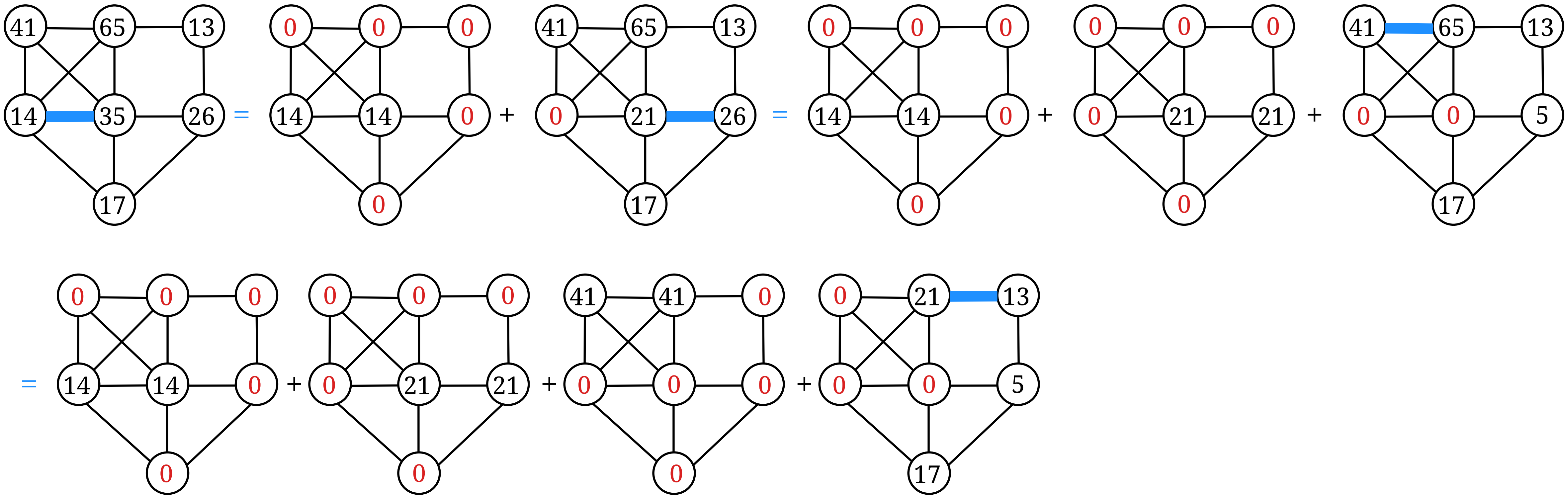
Example – continued



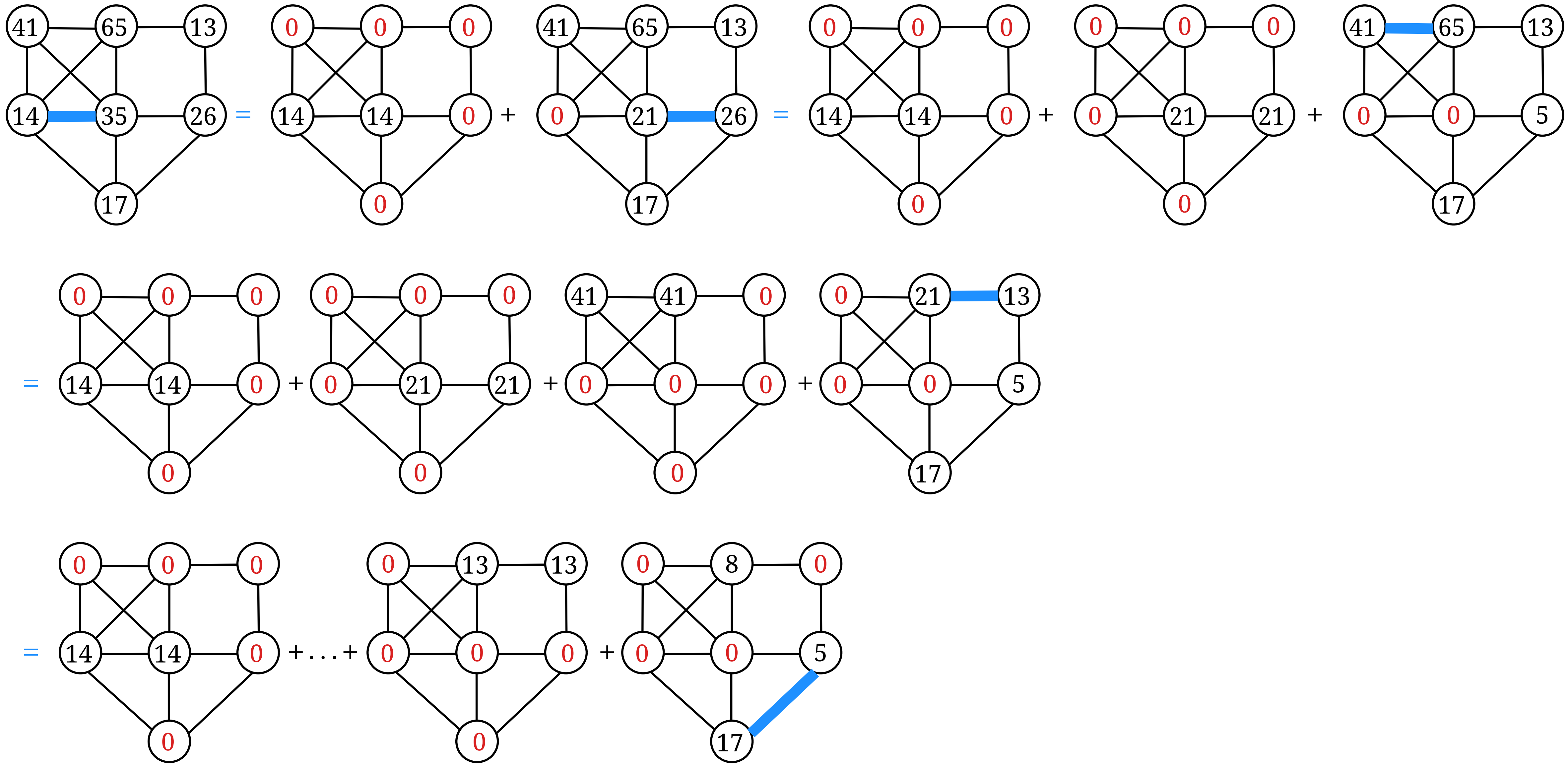
Example – continued



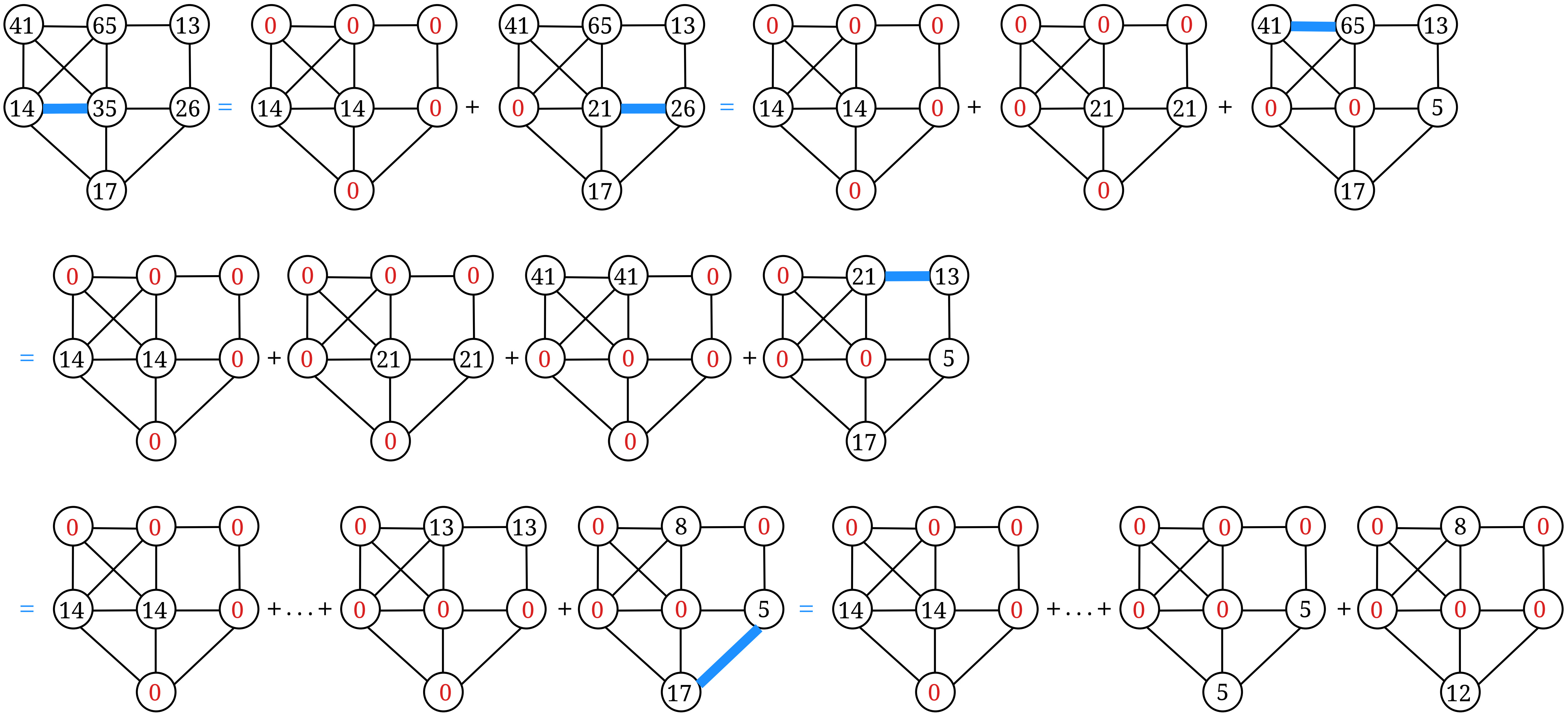
Example – continued



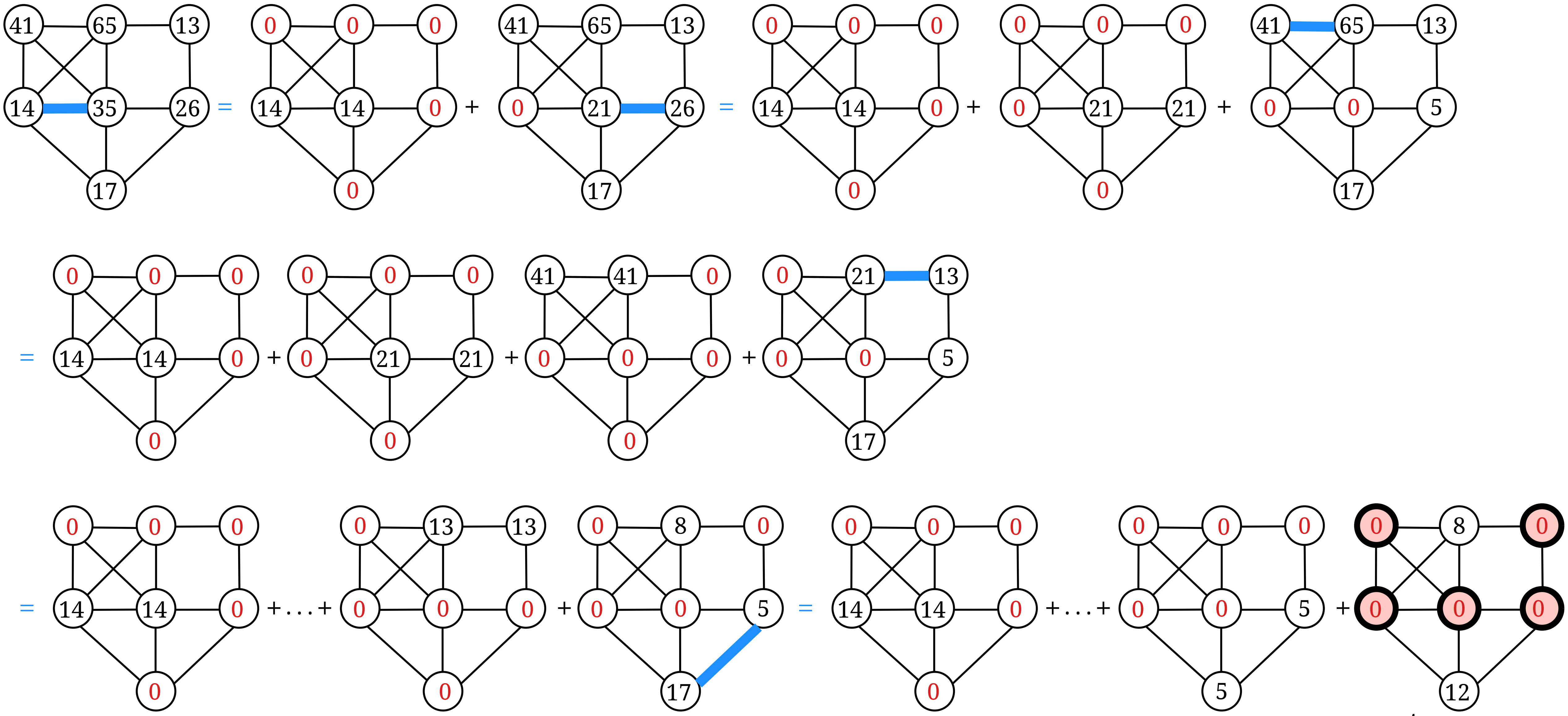
Example – continued



Example – continued

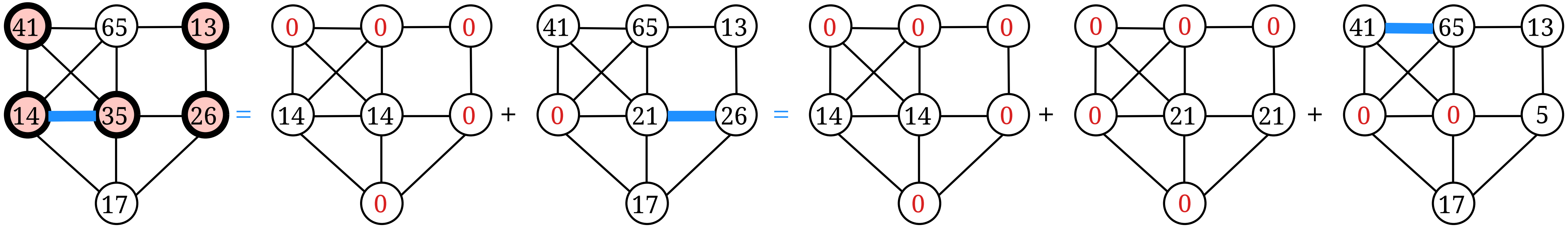


Example – continued

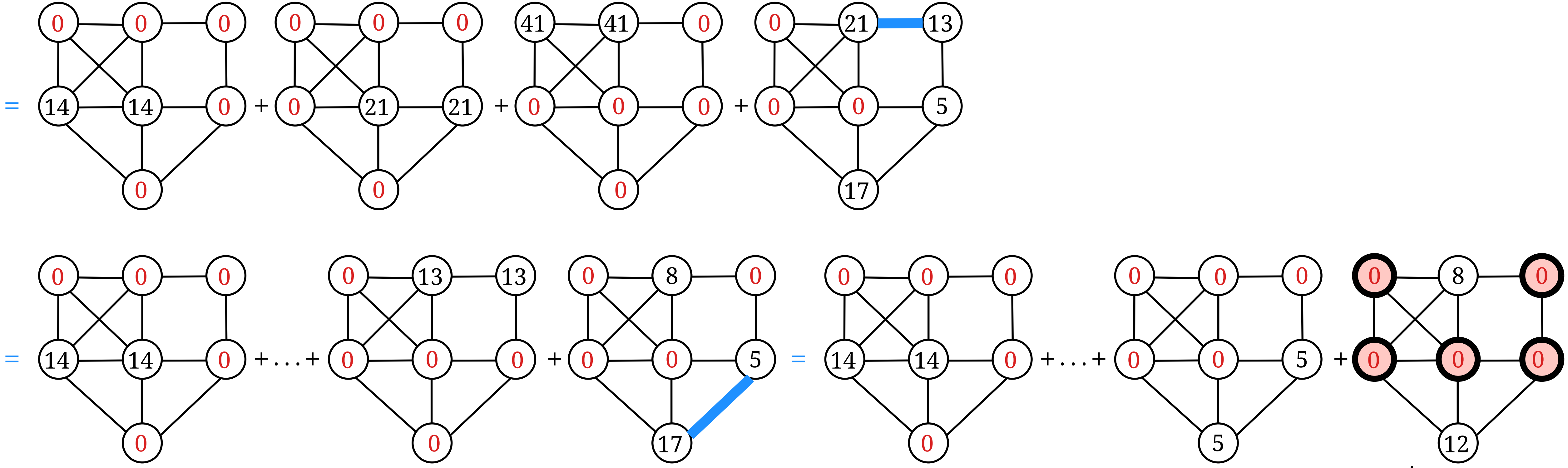


vertex cover of
weight 0

Example – continued

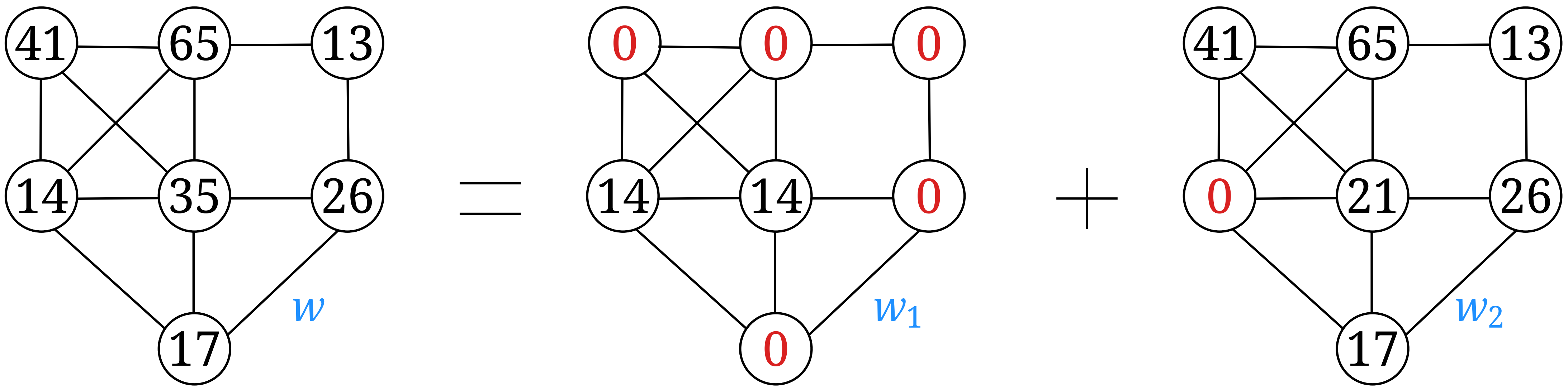


2-approximation



vertex cover of weight 0

Technique: Layering / Local Ratio

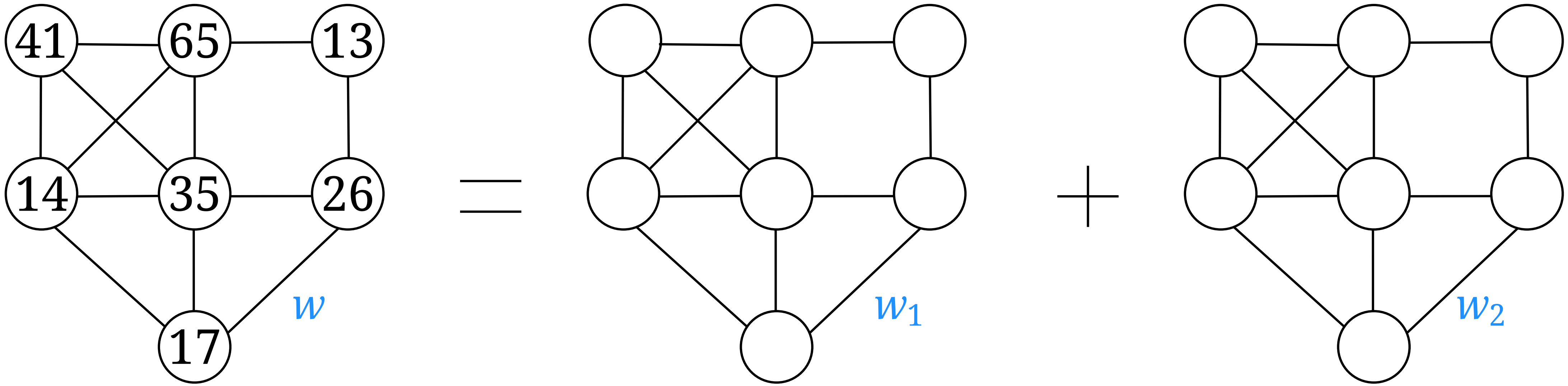


decompose weights: $w = w_1 + w_2$, and
consider instances I_w, I_{w_1}, and I_{w_2}

such that any feasible solution is
an r-approximation for w_1

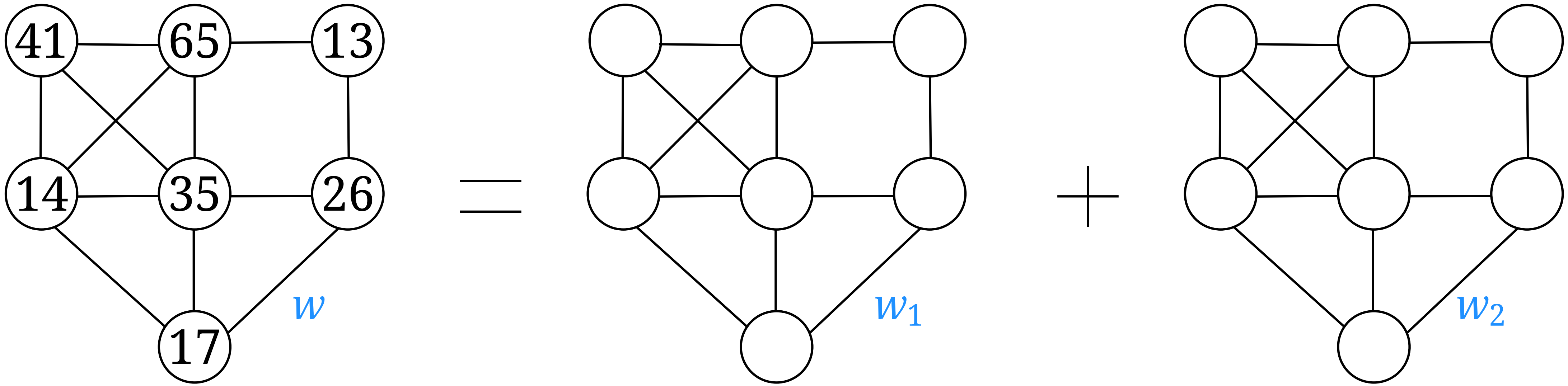
and recurse on w_2

Technique: Layering / Local Ratio



other choices of w_1 work too:

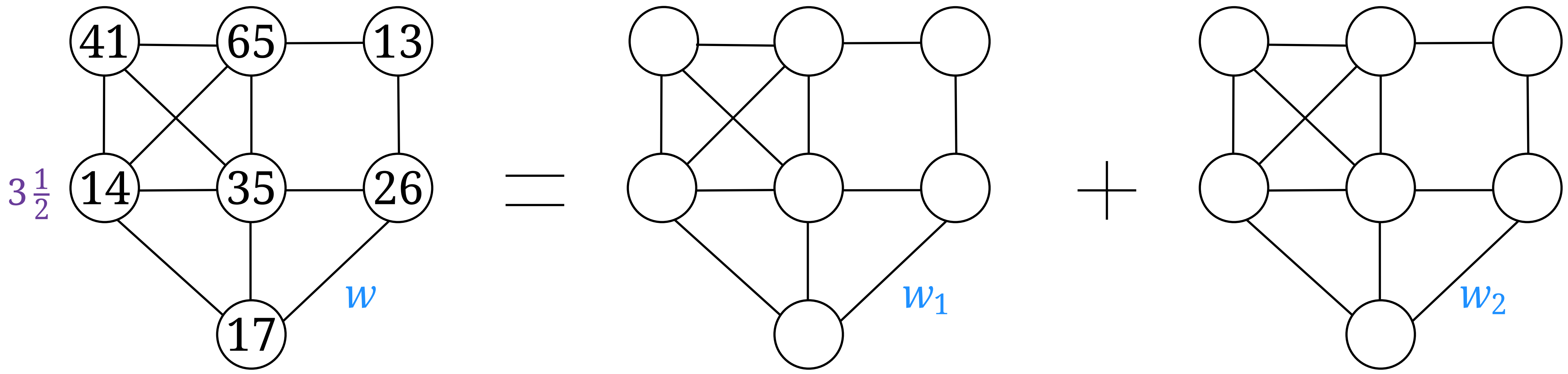
Technique: Layering / Local Ratio



other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

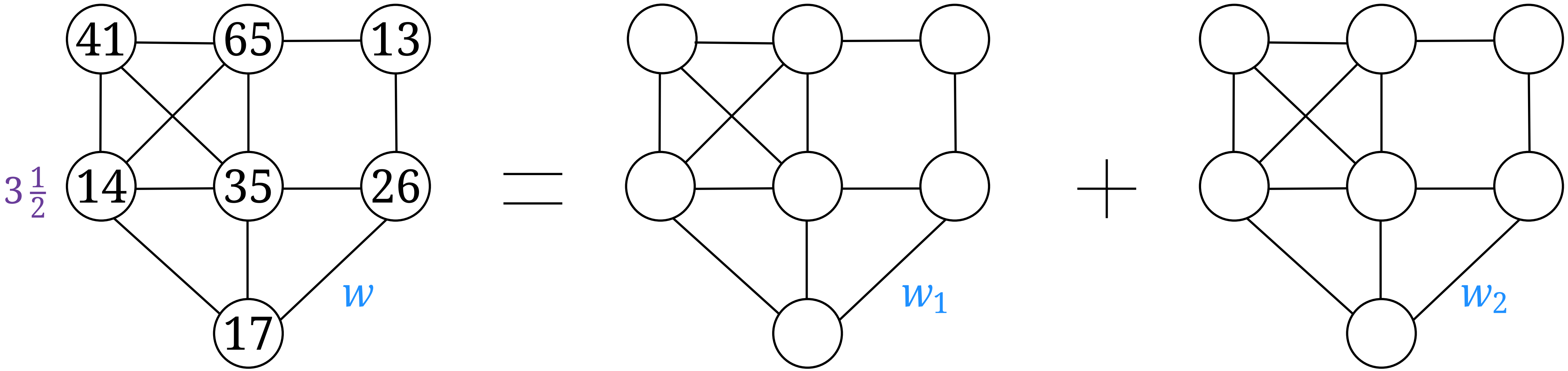
Technique: Layering / Local Ratio



other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

Technique: Layering / Local Ratio

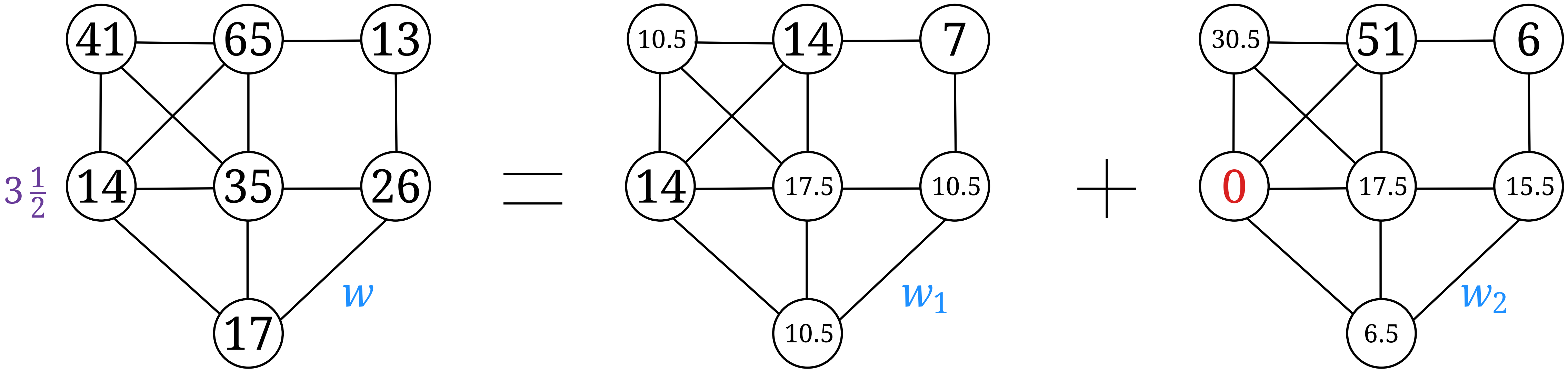


other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

$w_1(v) := c \cdot \text{degree}(v)$

Technique: Layering / Local Ratio

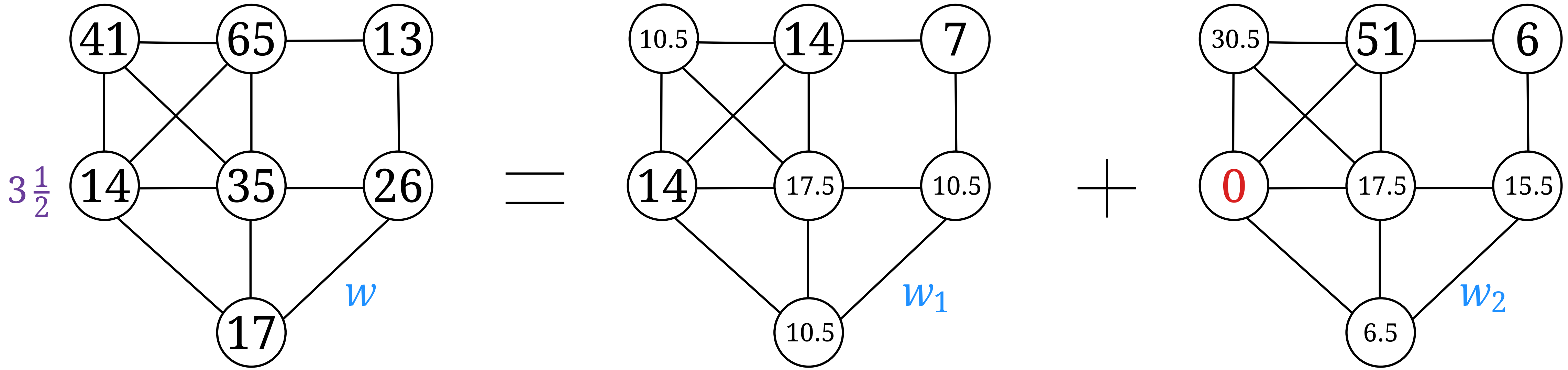


other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

$w_1(v) := c \cdot \text{degree}(v)$

Technique: Layering / Local Ratio



other choices of w_1 work too:

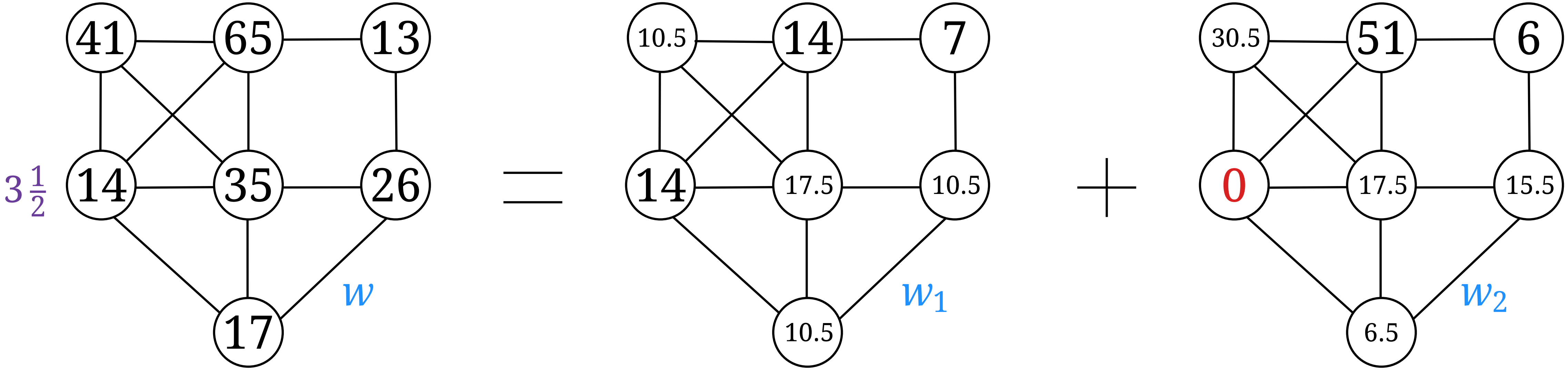
$c := \min(w(u)/\text{degree}(u), u \in V)$

$w_1(v) := c \cdot \text{degree}(v)$

before computing c:

- remove degree 0 nodes
  (not useful in cover)
- remove weight 0 nodes
  (and add to cover)

Technique: Layering / Local Ratio



other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

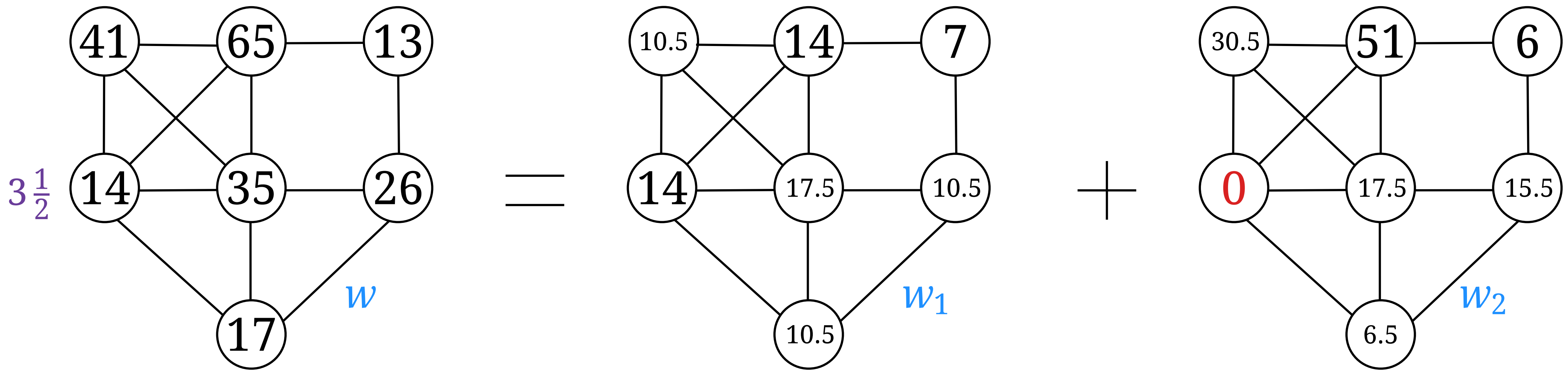
$w_1(v) := c \cdot \text{degree}(v)$

$w_1(V) = c \cdot \sum_{v \in V} \text{degree(v)}$

before computing c:

- remove degree 0 nodes
  (not useful in cover)
- remove weight 0 nodes
  (and add to cover)

Technique: Layering / Local Ratio



other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

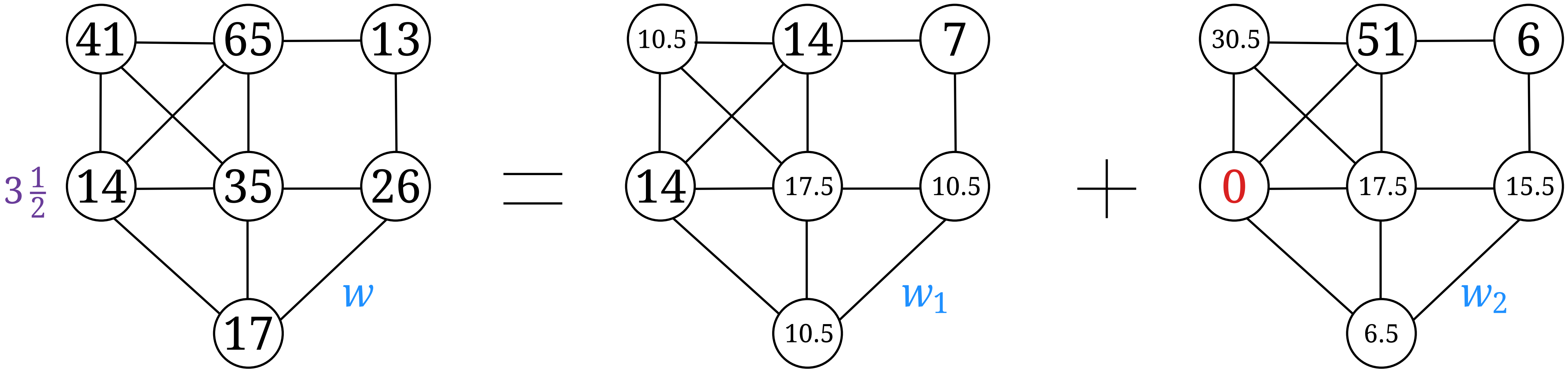
$w_1(v) := c \cdot \text{degree}(v)$

before computing c:

- remove degree 0 nodes (not useful in cover)
- remove weight 0 nodes (and add to cover)

$w_1(V) = c \cdot \sum_{v \in V} \text{degree(v)} \;\; = c \cdot 2|E|$ (handshaking lemma)

Technique: Layering / Local Ratio



other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

$w_1(v) := c \cdot \text{degree}(v)$

before computing c:

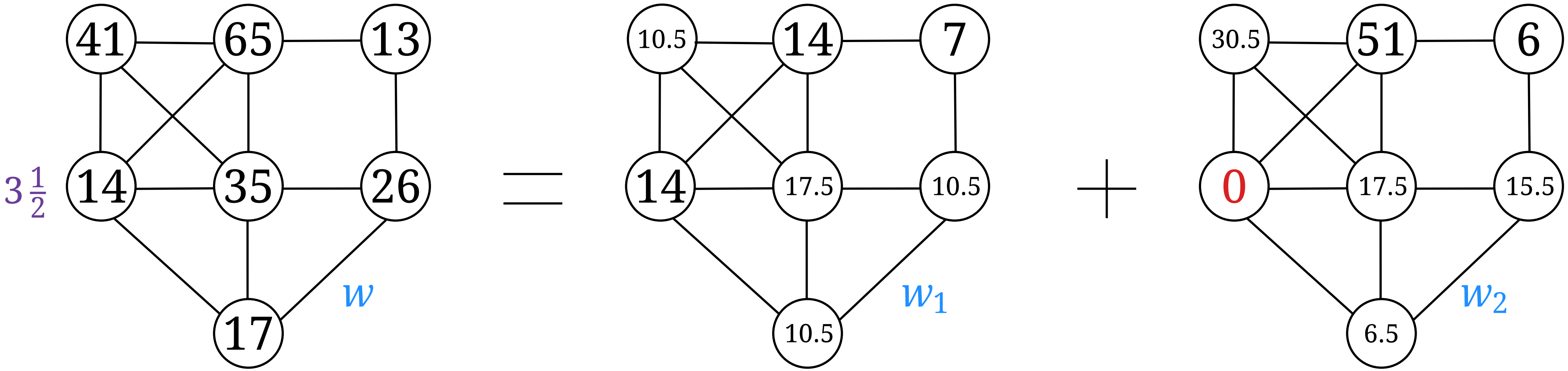
- remove degree 0 nodes (not useful in cover)
- remove weight 0 nodes (and add to cover)

$w_1(V) = c \cdot \sum_{v \in V} \text{degree(v)} \;\; = c \cdot 2|E|$ (handshaking lemma)

optimal vertex cover V_1^* has to cover all edges

$\Rightarrow |E| \leq \sum_{v \in V_1^*} \text{degree(v)}$

Technique: Layering / Local Ratio



other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

$w_1(v) := c \cdot \text{degree}(v)$

before computing c:

- remove degree 0 nodes (not useful in cover)
- remove weight 0 nodes (and add to cover)

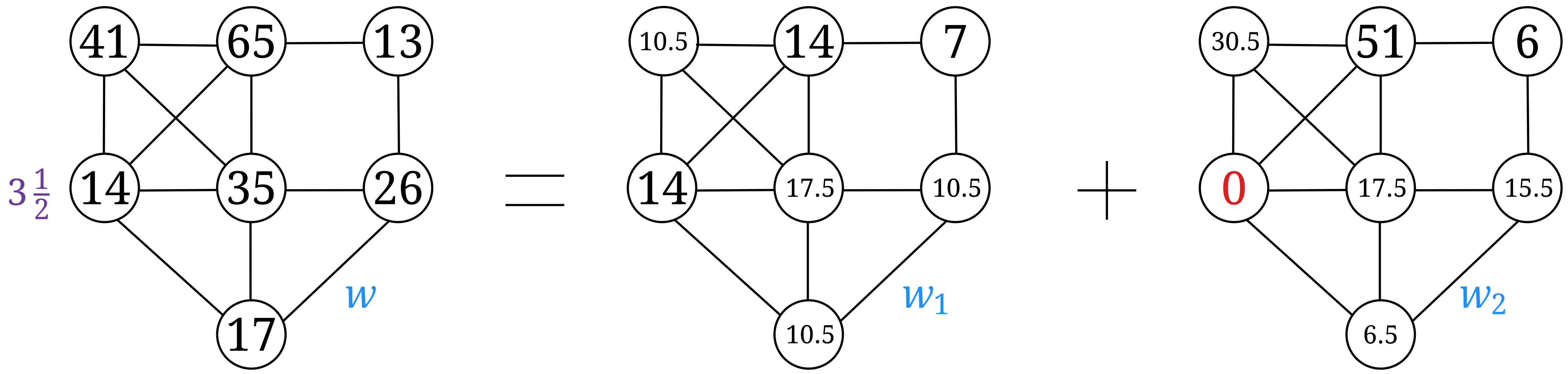
$w_1(V) = c \cdot \sum_{v \in V} \text{degree(v)} \;\; = c \cdot 2|E|$ (handshaking lemma)

optimal vertex cover V_1^* has to cover all edges

$\Rightarrow |E| \leq \sum_{v \in V_1^*} \text{degree(v)}$

$\Rightarrow w_1(V) \leq 2 \sum_{v \in V_1^*} c \cdot \text{degree(v)} = 2w_1(V_1^*)$

Technique: Layering / Local Ratio



other choices of w_1 work too:

$c := \min(w(u)/\text{degree}(u), u \in V)$

$w_1(v) := c \cdot \text{degree}(v)$

before computing c:

- remove degree 0 nodes (not useful in cover)
- remove weight 0 nodes (and add to cover)

$w_1(V) = c \cdot \sum_{v \in V} \text{degree(v)} \ = c \cdot 2|E|$ (handshaking lemma)

optimal vertex cover V_1^* has to cover all edges

$\Rightarrow |E| \leq \sum_{v \in V_1^*} \text{degree(v)}$

$\Rightarrow w_1(V) \leq 2 \sum_{v \in V_1^*} c \cdot \text{degree(v)} = 2 w_1(V_1^*)$

$\Rightarrow$ 2-approximation, generalizes to k-approximation for set cover with every element occuring in at most k sets.

Greedy SetCover applied to ShortestSuperString

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$

Q: What is the shortest superstring?

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$ $\longrightarrow cbaabcb\,?$

Q: What is the shortest superstring?

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$ $\longrightarrow cbaabcb\,?$

Q: What is the shortest superstring?

abc

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$ $\longrightarrow$ $cbaabcb$?

Q: What is the shortest superstring?

$$\begin{array}{l} abc \\ \;\; bcb \end{array}$$

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$ $\longrightarrow cbaabcb\,?$

Q: What is the shortest superstring?

abc
$\;\;bcb$
$\;\;\;\;cbaa$

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$ $\longrightarrow$ $cbaabcb\,?$

$abcbaa$

abc
$\;\;bcb$
$\;\;\;\;cbaa$

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$ $\longrightarrow$ $cbaabcb$?

$abcbaa$ “covers” all strings in U

abc

bcb

$cbaa$

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$ $\longrightarrow$ $cbaabcb$?

$abcbaa$ "covers" all strings in U

$$\begin{array}{l} abc \\ bcb \\ cbaa \end{array}$$

W.l.o.g.: No string s_i is a substring of any other string s_j.

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\}$

Greedy approach: iteratively combine two strings with maximum overlap

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\} \longrightarrow \{cbaa, abcb\}$

Greedy approach: iteratively combine two strings with maximum overlap

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\} \longrightarrow \{cbaa, abcb\} \longrightarrow \{abcbaa\}$

Greedy approach: iteratively combine two strings with maximum overlap

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\} \longrightarrow \{cbaa, abcb\} \longrightarrow \{abcbaa\}$

Greedy approach: iteratively combine two strings with maximum overlap

Interestingly, only approximation factor ≥ 2 is known for this algorithm.

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\} \longrightarrow \{cbaa, abcb\} \longrightarrow \{abcbaa\}$

Greedy approach: iteratively combine two strings with maximum overlap

Interestingly, only approximation factor ≥ 2 is known for this algorithm.

Q: Example where approximation factor 2 is achieved?

ShortestSuperString (SSS)

Given a set $\{s_1, \ldots, s_n\} \subseteq \Sigma^+$ of strings over a finite alphabet Σ.

Find a **shortest string** s (superstring) such that,
for each $i \in \{1, \ldots, n\}$, the string s_i is a substring of s.

Example. $U := \{cbaa, abc, bcb\} \longrightarrow \{cbaa, abcb\} \longrightarrow \{abcbaa\}$

Greedy approach: iteratively combine two strings with maximum overlap

Interestingly, only approximation factor ≥ 2 is known for this algorithm.

Q: Example where approximation factor 2 is achieved?

$U' := \{ab^k, b^k c, b^{k+1}\} \longrightarrow \{ab^k c, b^{k+1}\} \longrightarrow \{ab^k cb^{k+1}\}$

SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

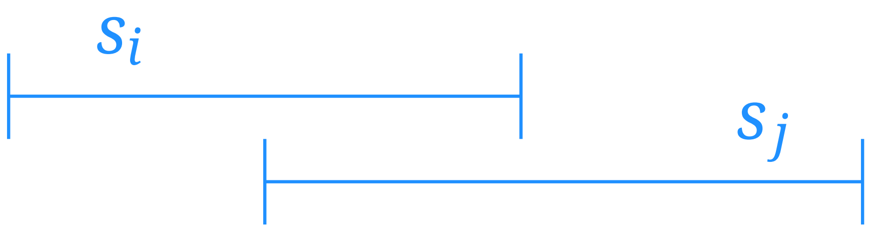
Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

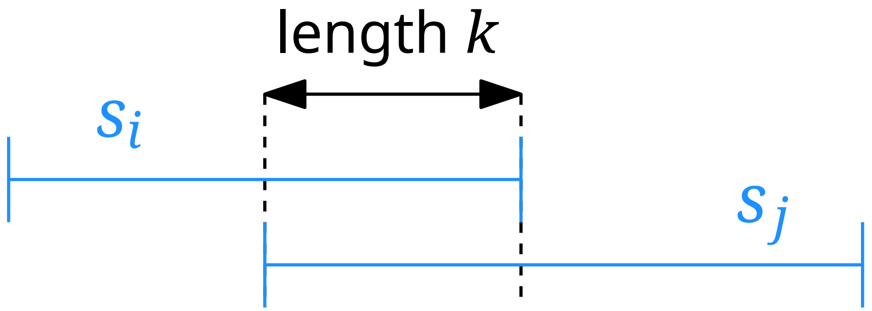


SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

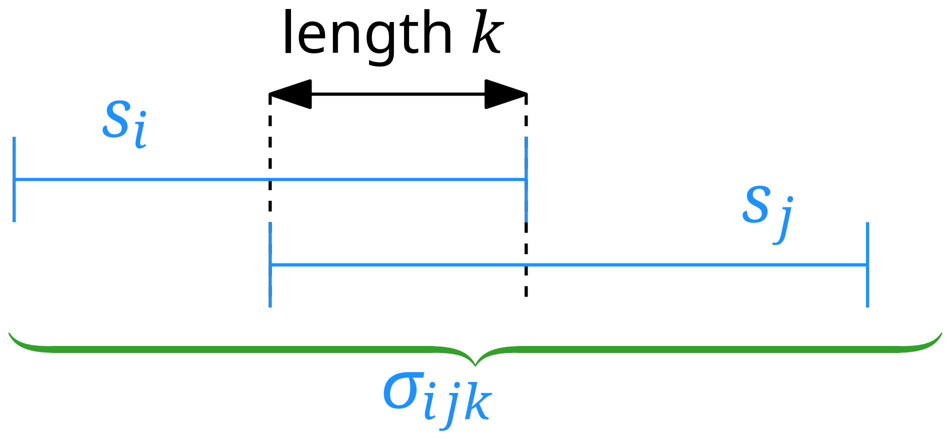


SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)



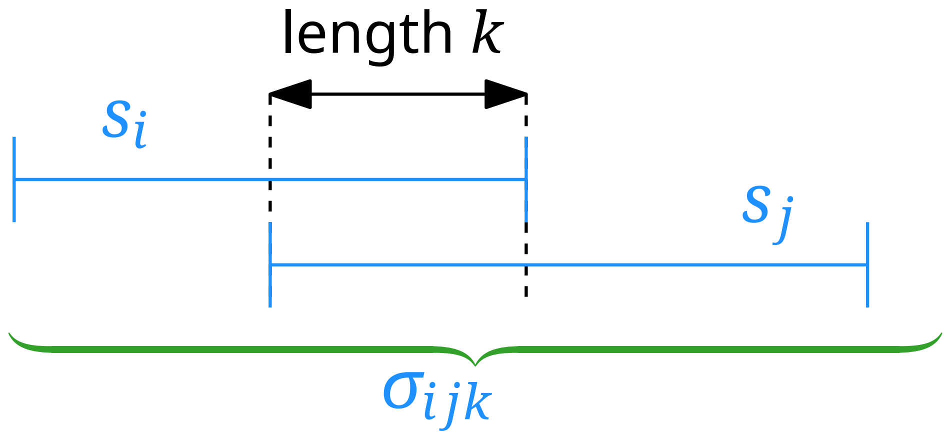
SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

s_i: cabab s_j: ababc



SSS as a SETCOVER Problem

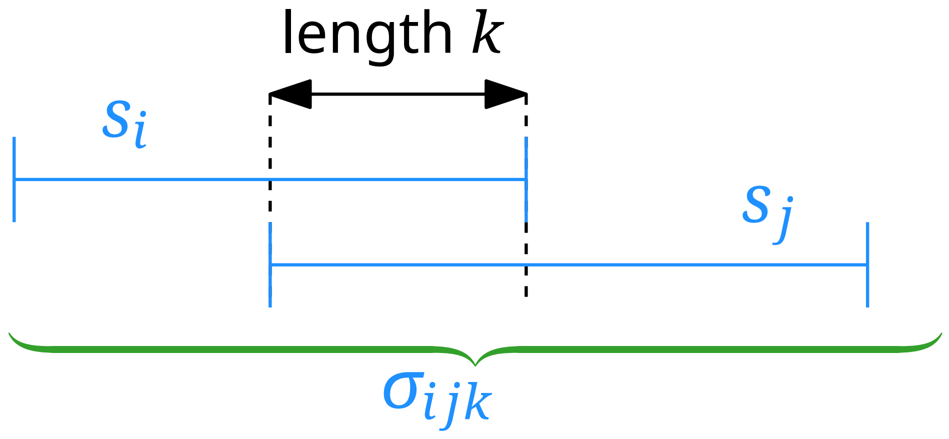
SETCOVER Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

s_i: cabab s_j: ababc

cabab
  ababc



SSS as a SetCover Problem

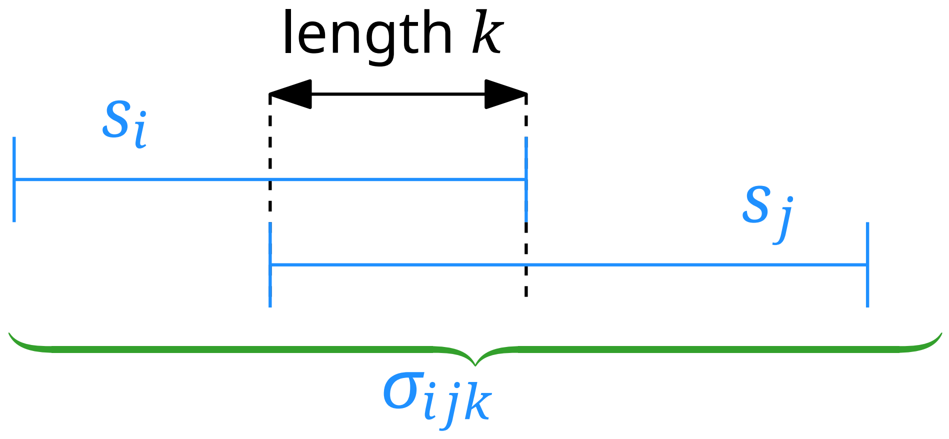
SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

s_i: cabab s_j: ababc

cabab
  ababc



SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

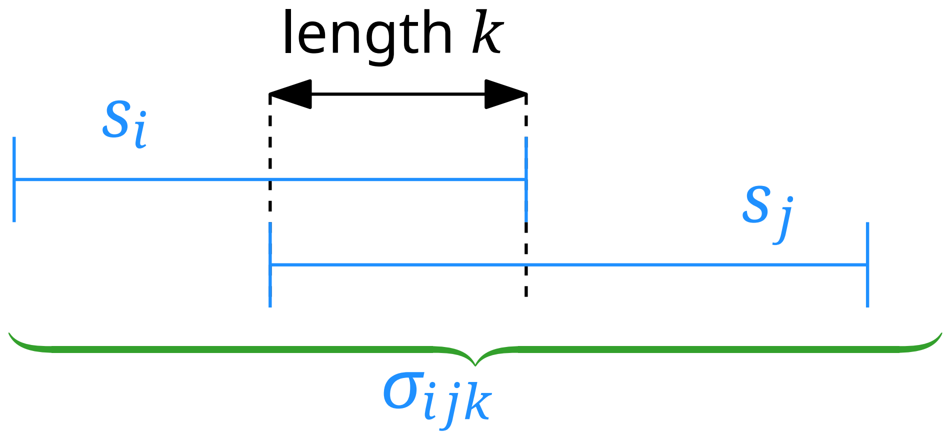
Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

s_i: cabab s_j: ababc

cabab
ababc

σ_{ij2}: cabababc



SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

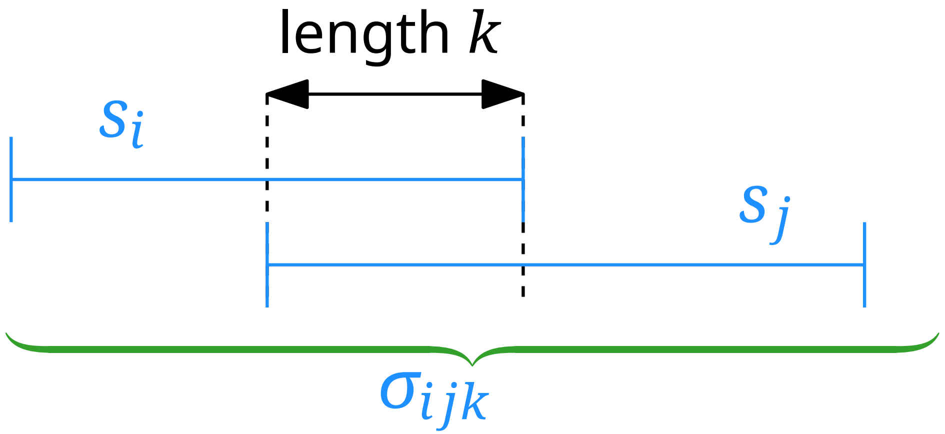
Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

s_i: cabab s_j: ababc

```
cabab          cabab
   ababc        ababc
```

σ_{ij2}: cabababc



SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

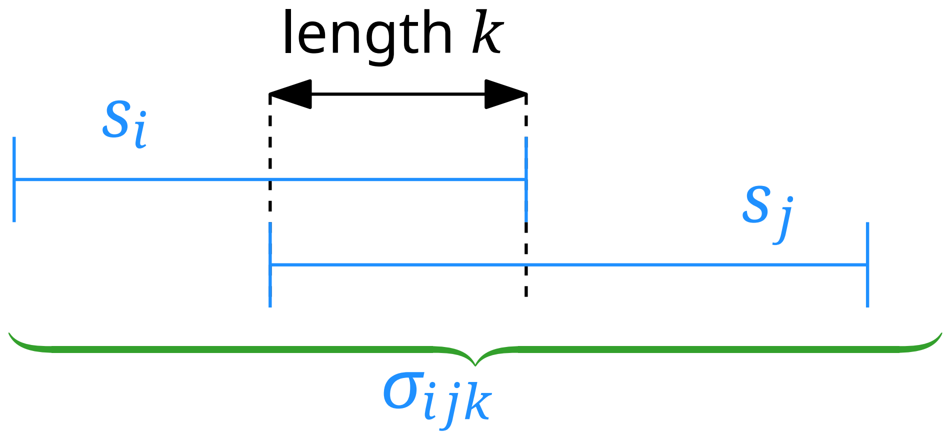
s_i: cabab s_j: ababc

cabab
  ababc

σ_{ij2}: cababab c

cabab
 ababc

σ_{ij4}: cababc



SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

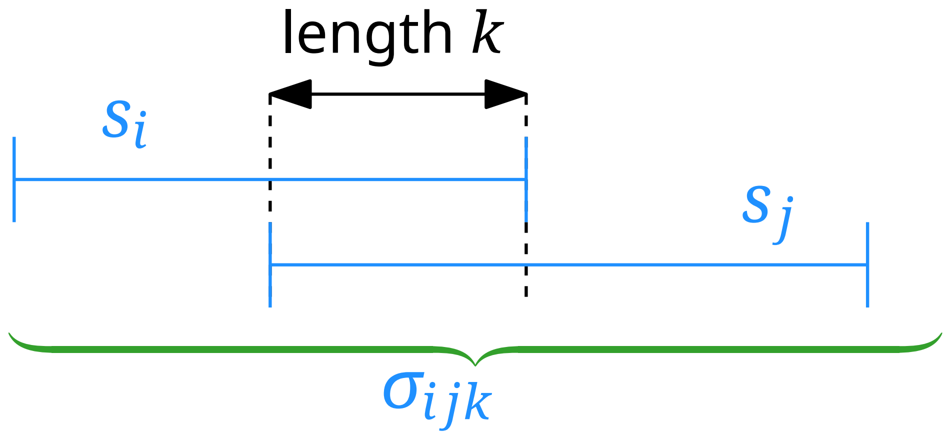
Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

s_i: cabab s_j: ababc

```
cabab            cabab
   ababc          ababc
```

σ_{ij2}: cabababc σ_{ij4}: cababc



$S(\sigma_{ijk}) = \{s \in U \mid s \text{ substring of } \sigma_{ijk}\}$ contains the elements of the ground set covered by σ_{ijk}.

SSS as a SETCOVER Problem

SETCOVER Instance: ground set U, set family $\mathcal{S}$, costs c.

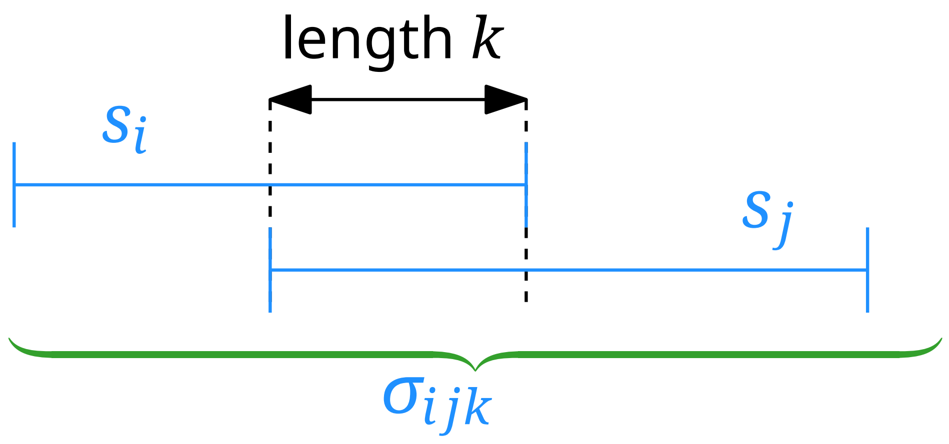
Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

s_i: cabab s_j: ababc

```
cabab        cabab
  ababc       ababc
```

σ_{ij2}: cababab c σ_{ij4}: cababc



$S(\sigma_{ijk}) = \{s \in U \mid s \text{ substring of } \sigma_{ijk}\}$ contains the elements of the ground set covered by σ_{ijk}.

costs $c\left(S(\sigma_{ijk})\right) = |\sigma_{ijk}|$ (number of characters in σ_{ijk})

SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

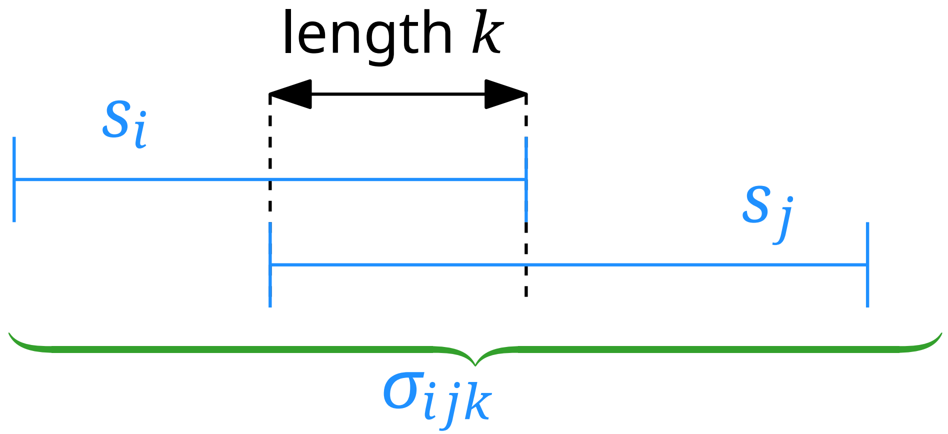
Ground set $U := \{s_1, \ldots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

s_i: cabab s_j: ababc

cabab
ababc
σ_{ij2}: cababababc

cabab
ababc
σ_{ij4}: cababc



$S(\sigma_{ijk}) = \{s \in U \mid s \text{ substring of } \sigma_{ijk}\}$ contains the elements of the ground set covered by σ_{ijk}.

costs $c\left(S(\sigma_{ijk})\right) = |\sigma_{ijk}|$ (number of characters in σ_{ijk})

set family $\mathcal{S} = \{S(\sigma_{ijk}) \mid k > 0\}$ (possibly $i = j$)

SSS as a SetCover Problem

SetCover Instance: ground set U, set family $\mathcal{S}$, costs c.

Ground set $U := \{s_1, \dots, s_n\}$.

Let be σ_{ijk} be the unique string with prefix s_i and suffix s_j where s_i and s_j overlap on k characters (for suitable i, j, k)

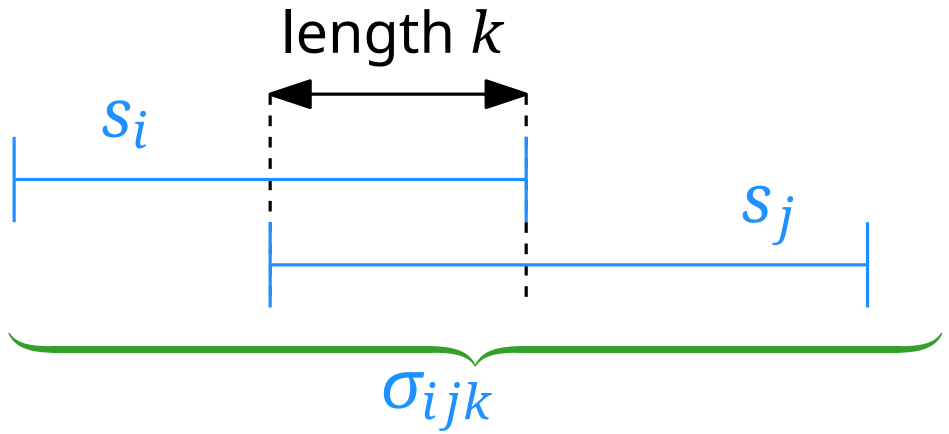
s_i: cabab s_j: ababc

cabab
   ababc

σ_{ij2}: cababababc

cabab
ababc

σ_{ij4}: cababc



$S(\sigma_{ijk}) = \{s \in U \mid s$ substring of $\sigma_{ijk}\}$ contains the elements of the ground set covered by σ_{ijk}.

costs $c\left(S(\sigma_{ijk})\right) = |\sigma_{ijk}|$ (number of characters in σ_{ijk})

set family $\mathcal{S} = \{S(\sigma_{ijk}) \mid k > 0\}$ (possibly $i = j$)

Relating SSS and SetCover

Lemma. Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of U, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

Relating SSS and SetCover

Lemma. Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of U, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

Proof.

Consider an optimal set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ of U.

Relating SSS and SetCover

Lemma. Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of U, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

Proof.

Consider an optimal set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ of U.

Then $s := \pi_1 \circ \cdots \circ \pi_k$ is a superstring of U of length

Relating SSS and SetCover

Lemma. Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of U, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

Proof.

Consider an optimal set cover $\{S(\pi_1), \dots, S(\pi_k)\}$ of U.

Then $s := \pi_1 \circ \dots \circ \pi_k$ is a superstring of U of length $\sum_{i=1}^{k} |\pi_i| = \sum_{i=1}^{k} c(S(\pi_i)) = \mathrm{OPT}_{\mathrm{SC}}$.

Relating SSS and SetCover

Lemma. Let $\mathrm{OPT}_{\mathrm{SSS}}$ be the length of a shortest superstring of U, and let $\mathrm{OPT}_{\mathrm{SC}}$ be the minimum cost of the corresponding SetCover instance. Then

$$\mathrm{OPT}_{\mathrm{SSS}} \leq \mathrm{OPT}_{\mathrm{SC}}.$$

Proof.

Consider an optimal set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ of U.

Then $s := \pi_1 \circ \cdots \circ \pi_k$ is a superstring of U of length $\sum_{i=1}^{k} |\pi_i| = \sum_{i=1}^{k} c(S(\pi_i)) = \mathrm{OPT}_{\mathrm{SC}}$.

Thus, $\mathrm{OPT}_{\mathrm{SSS}} \leq |s| = \mathrm{OPT}_{\mathrm{SC}}$.

Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof. Consider an optimal superstring s.

Relating SSS and SETCOVER

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof. Consider an optimal superstring s.

s

Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof. Consider an optimal superstring s.

Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

s

Relating SSS and SETCOVER

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

leftmost occurence of a string $s_{b_1} \in U$.

Relating SSS and SETCOVER

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

leftmost occurence of another string in U.

Relating SSS and SetCover

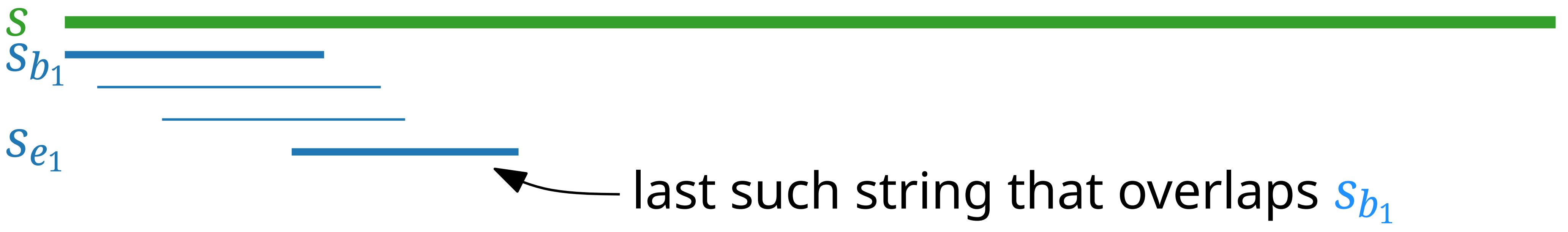
Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

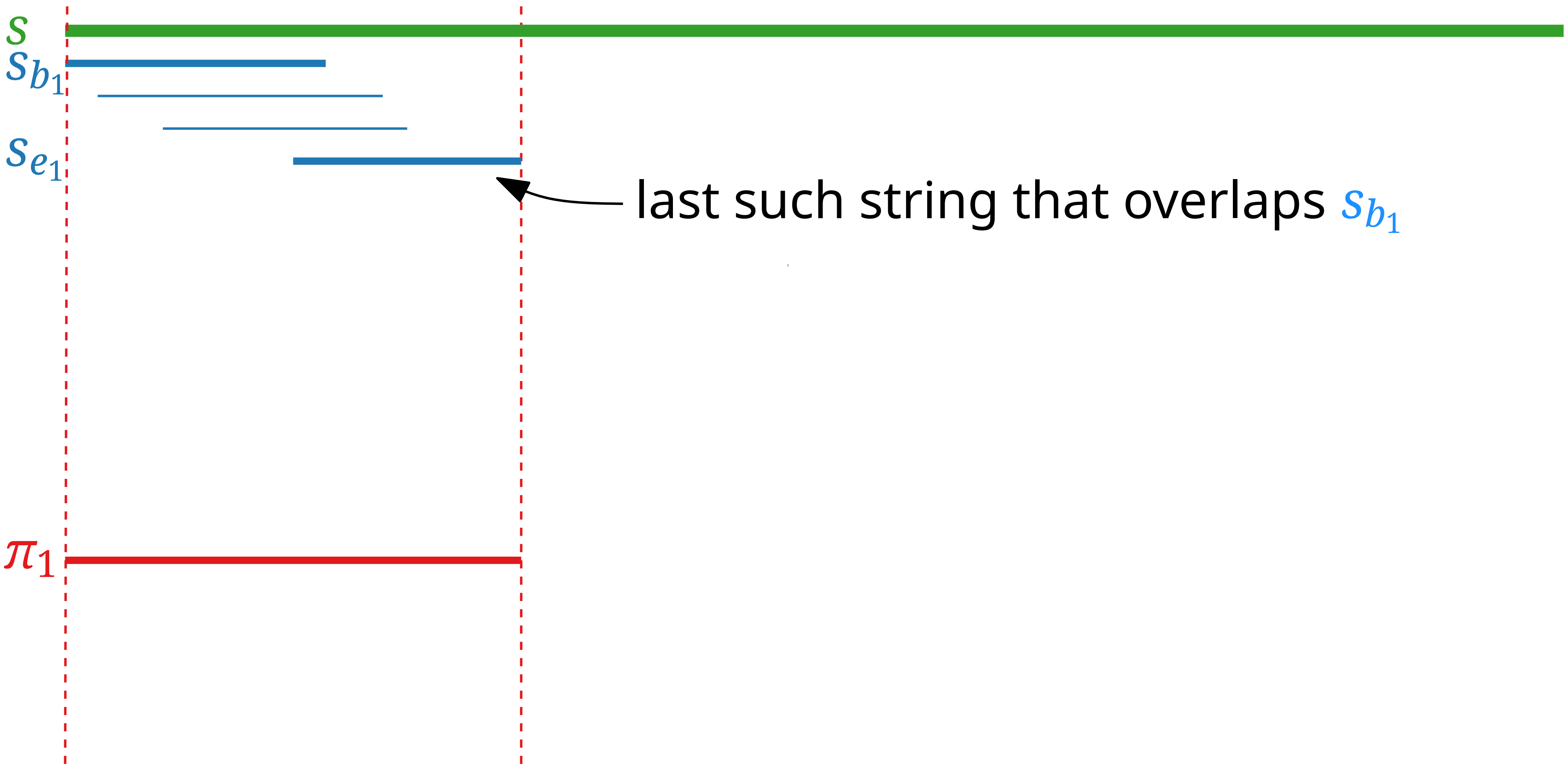
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

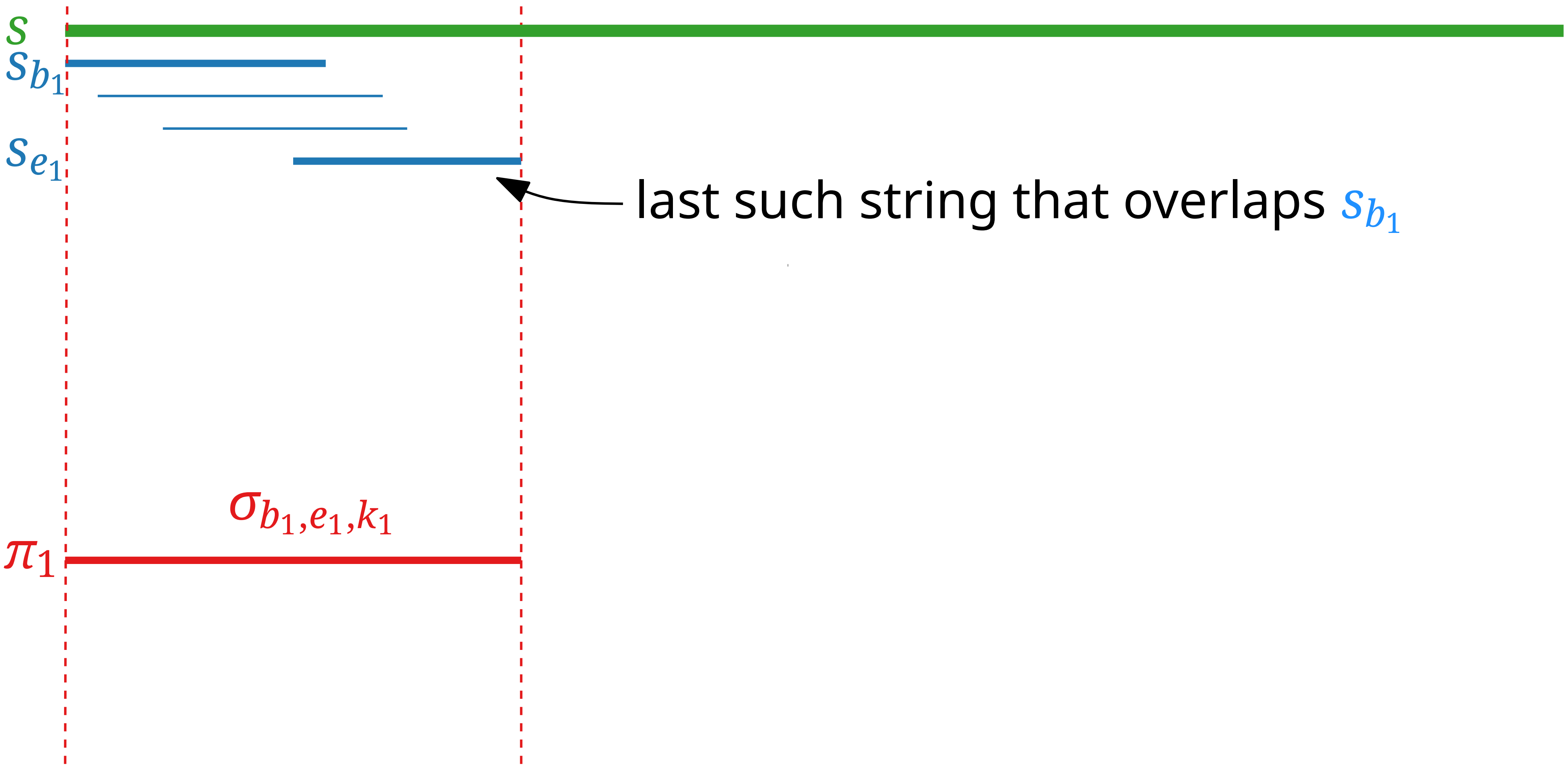
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

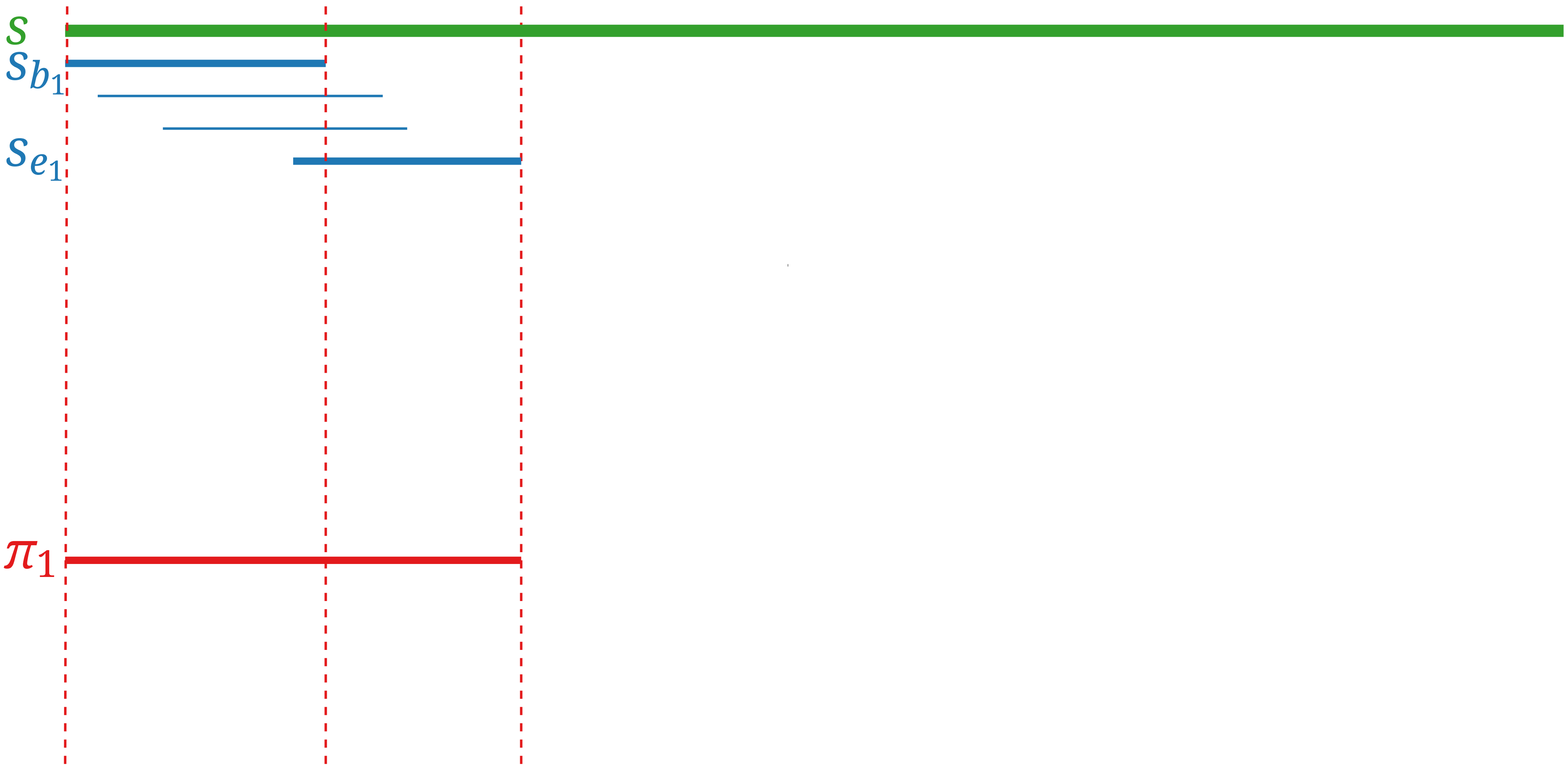
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

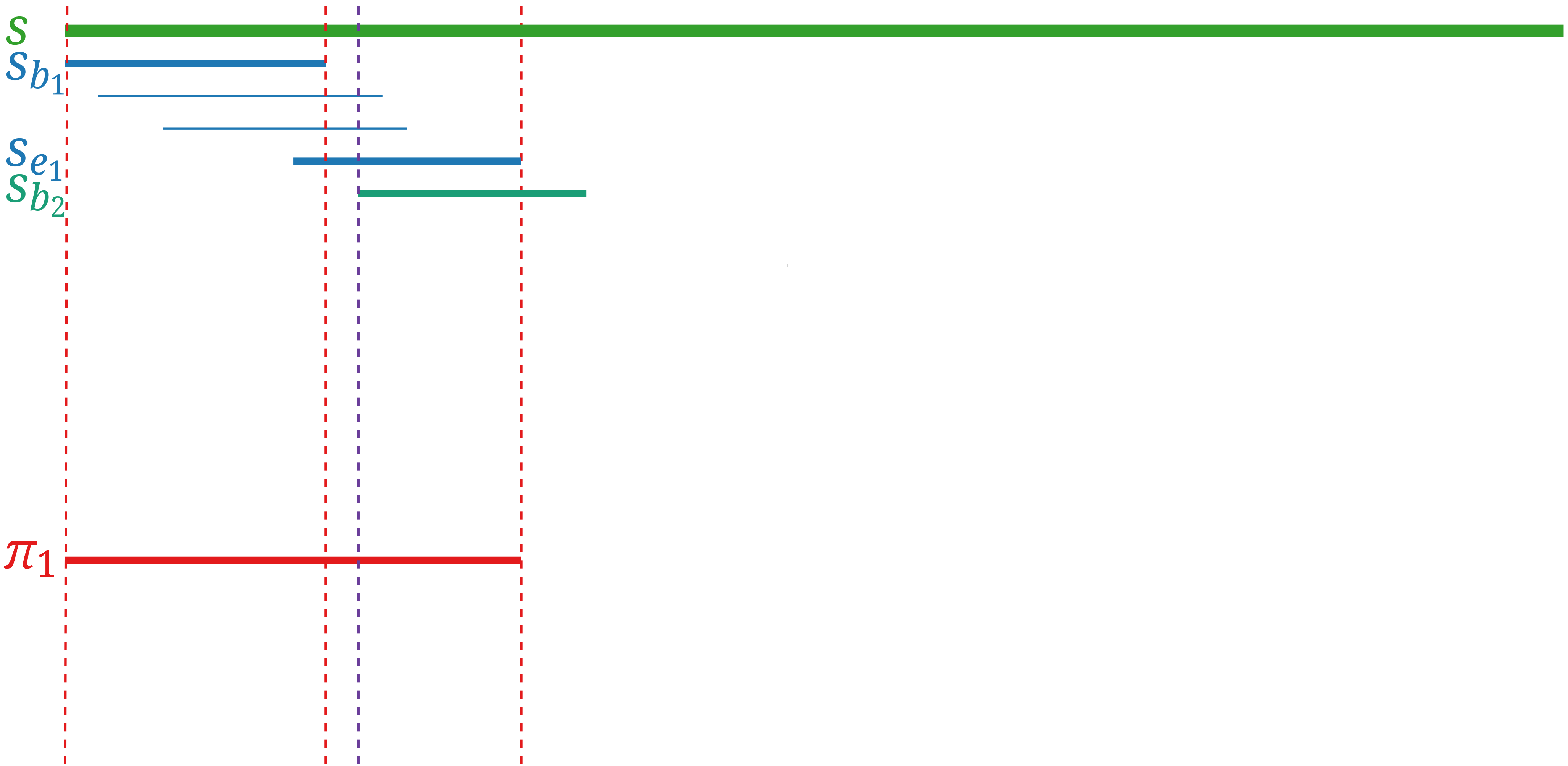
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

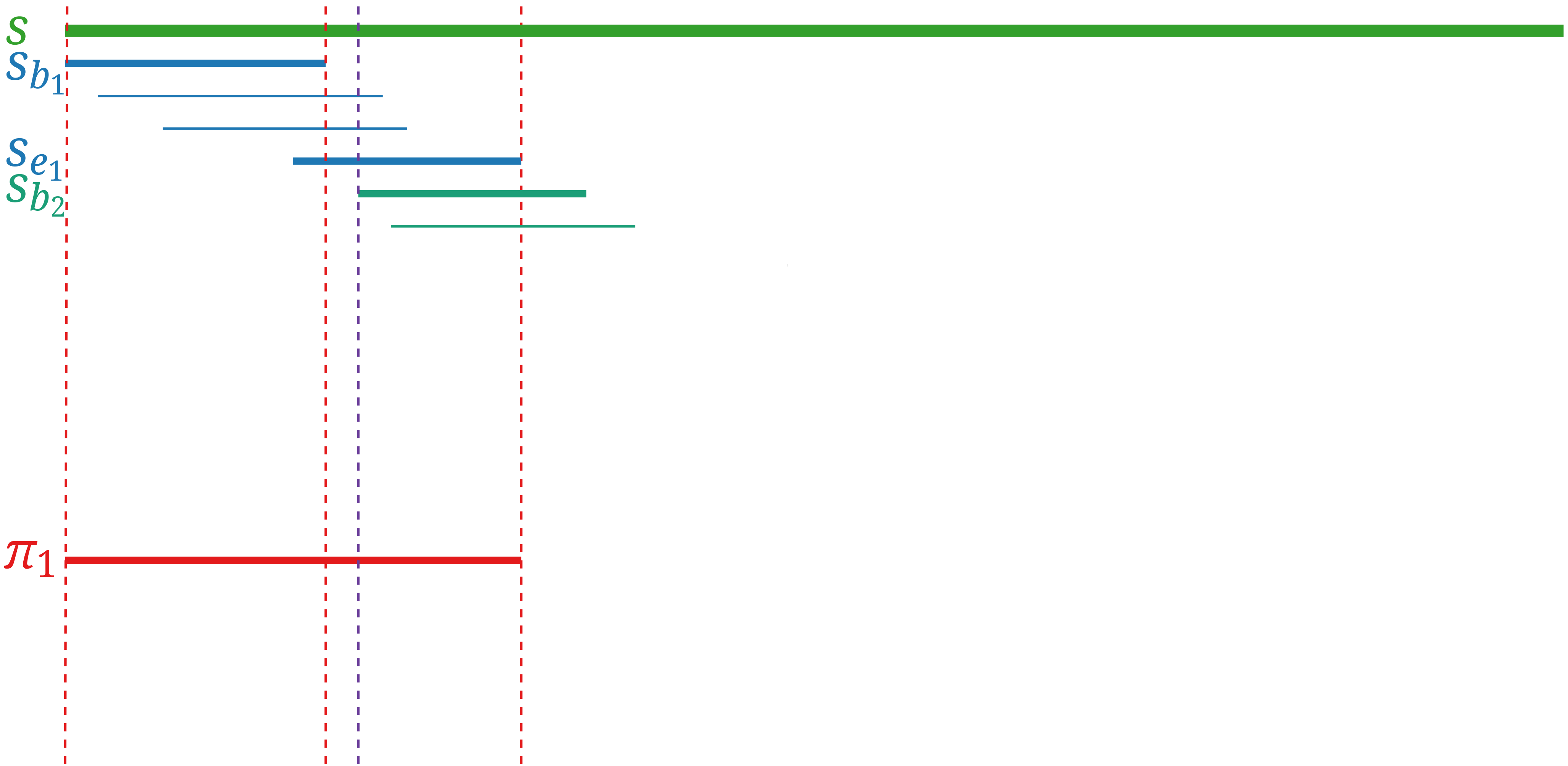
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

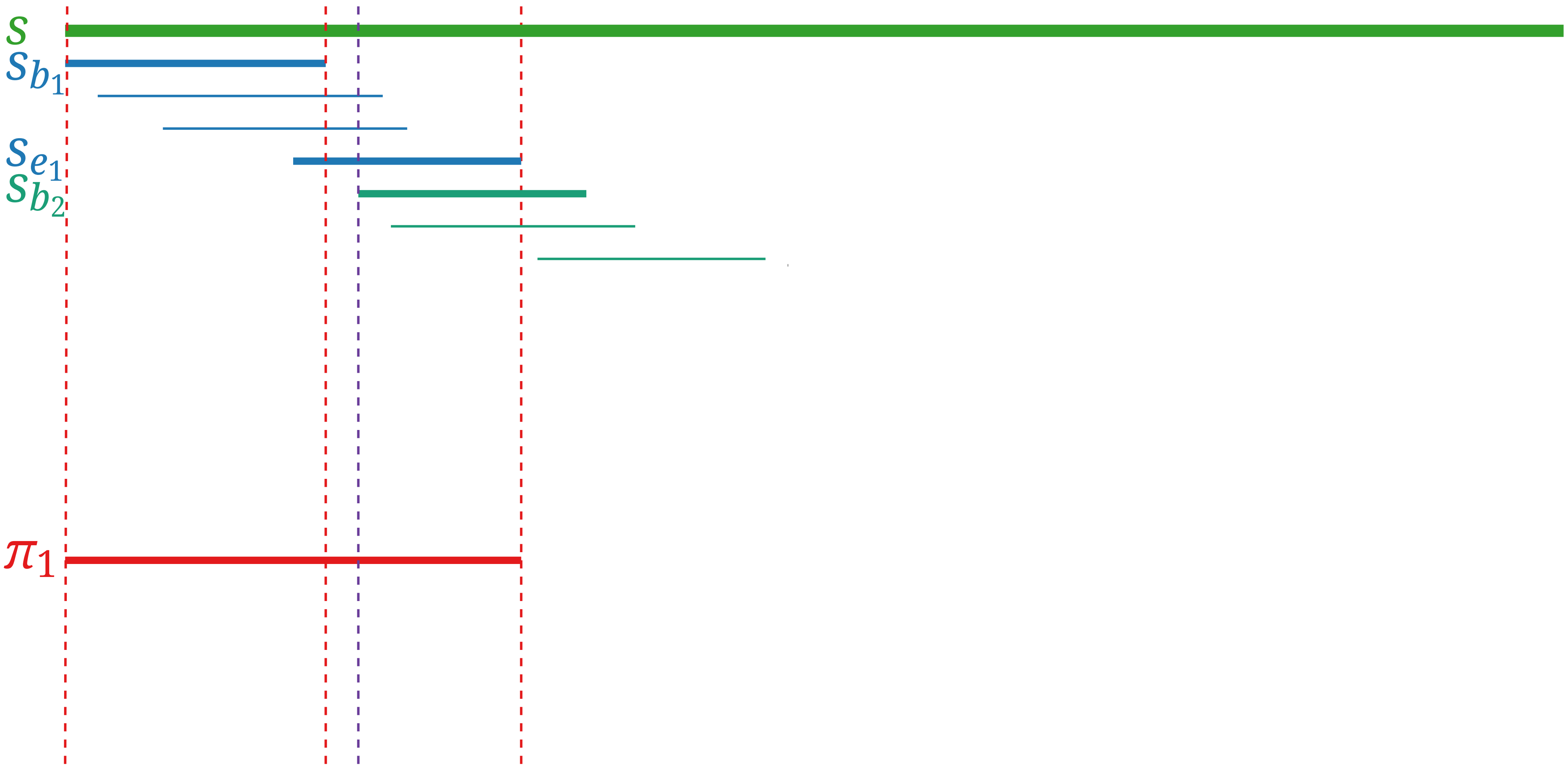
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

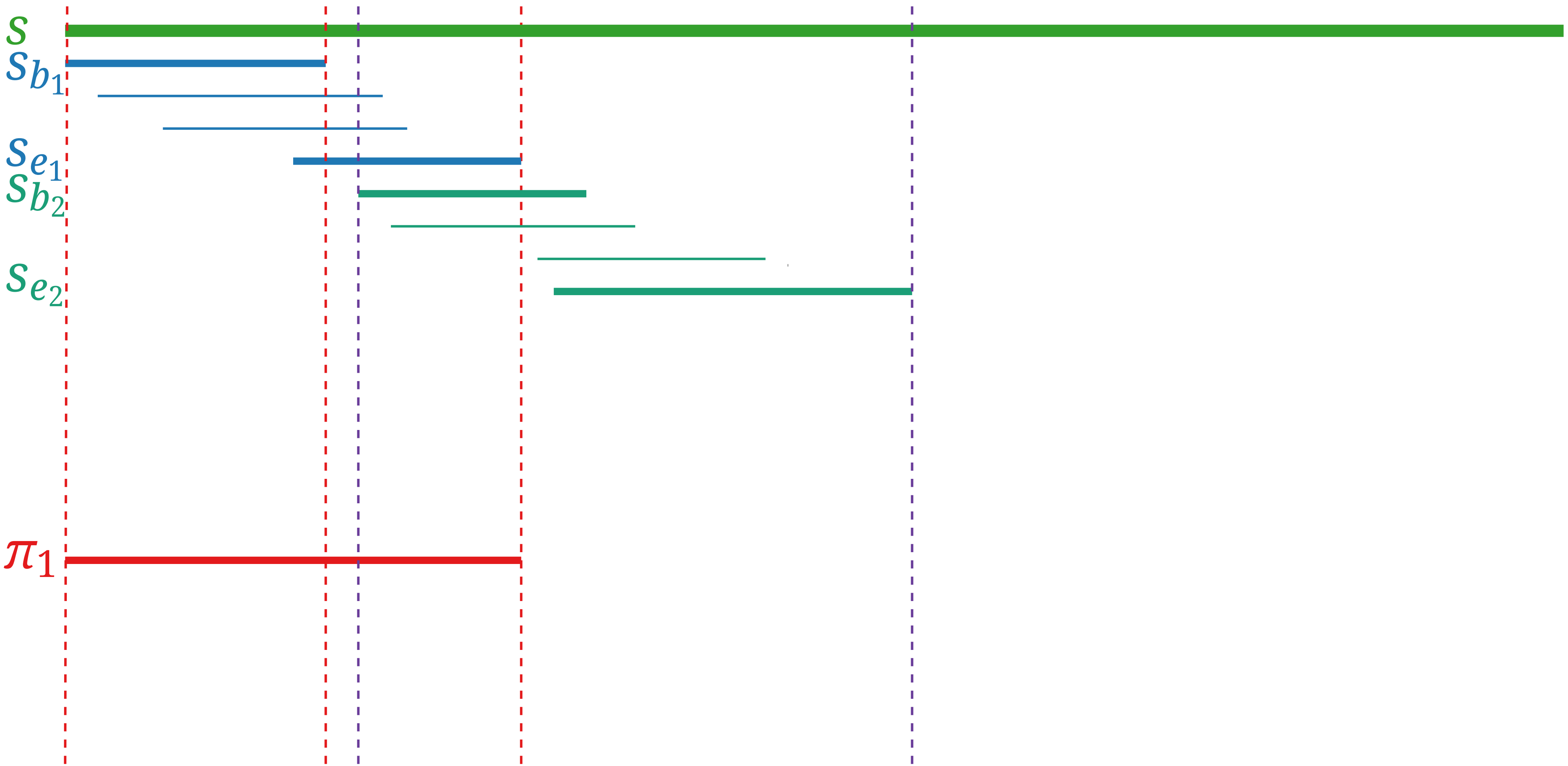
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

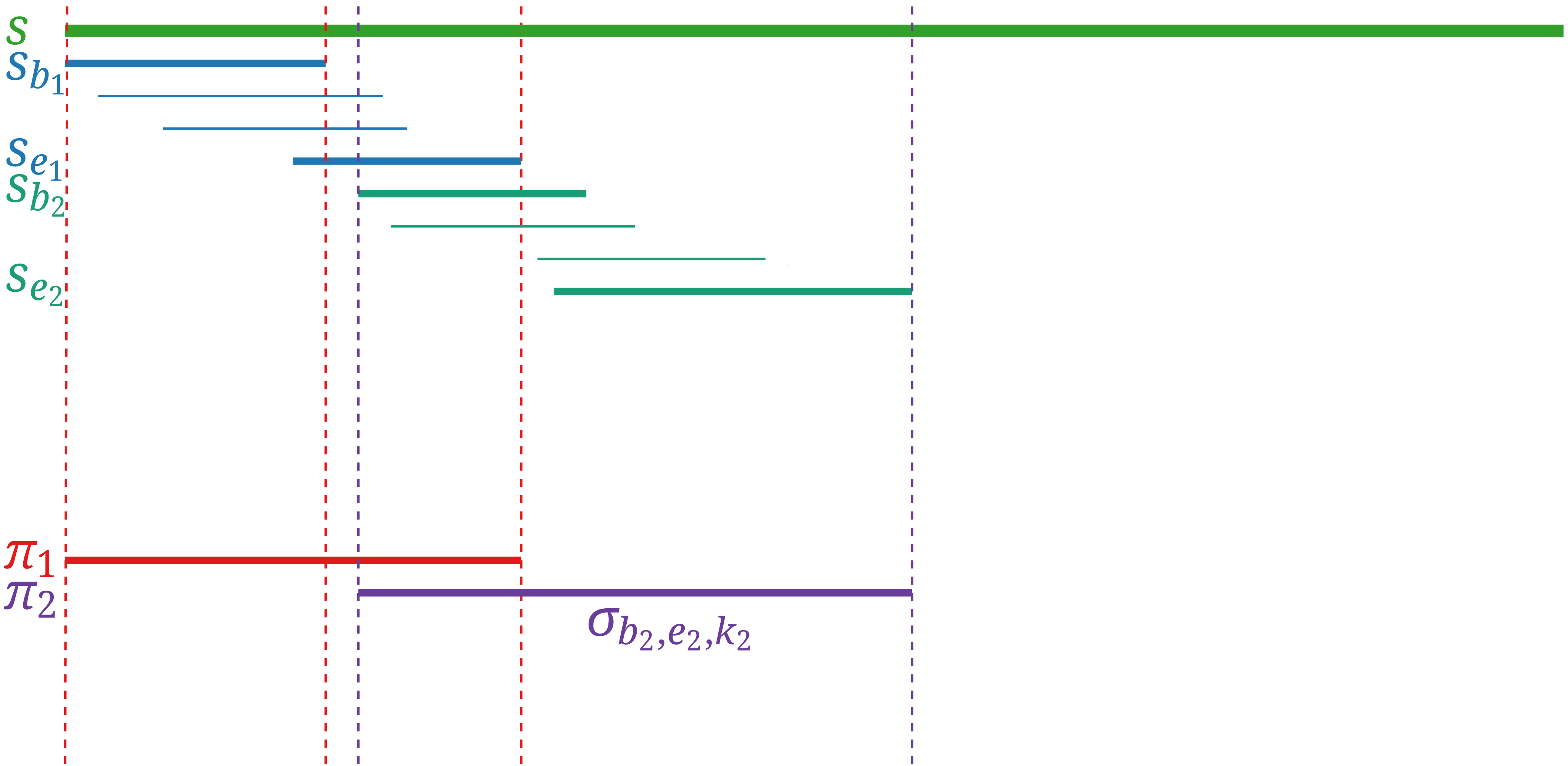
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SETCOVER

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

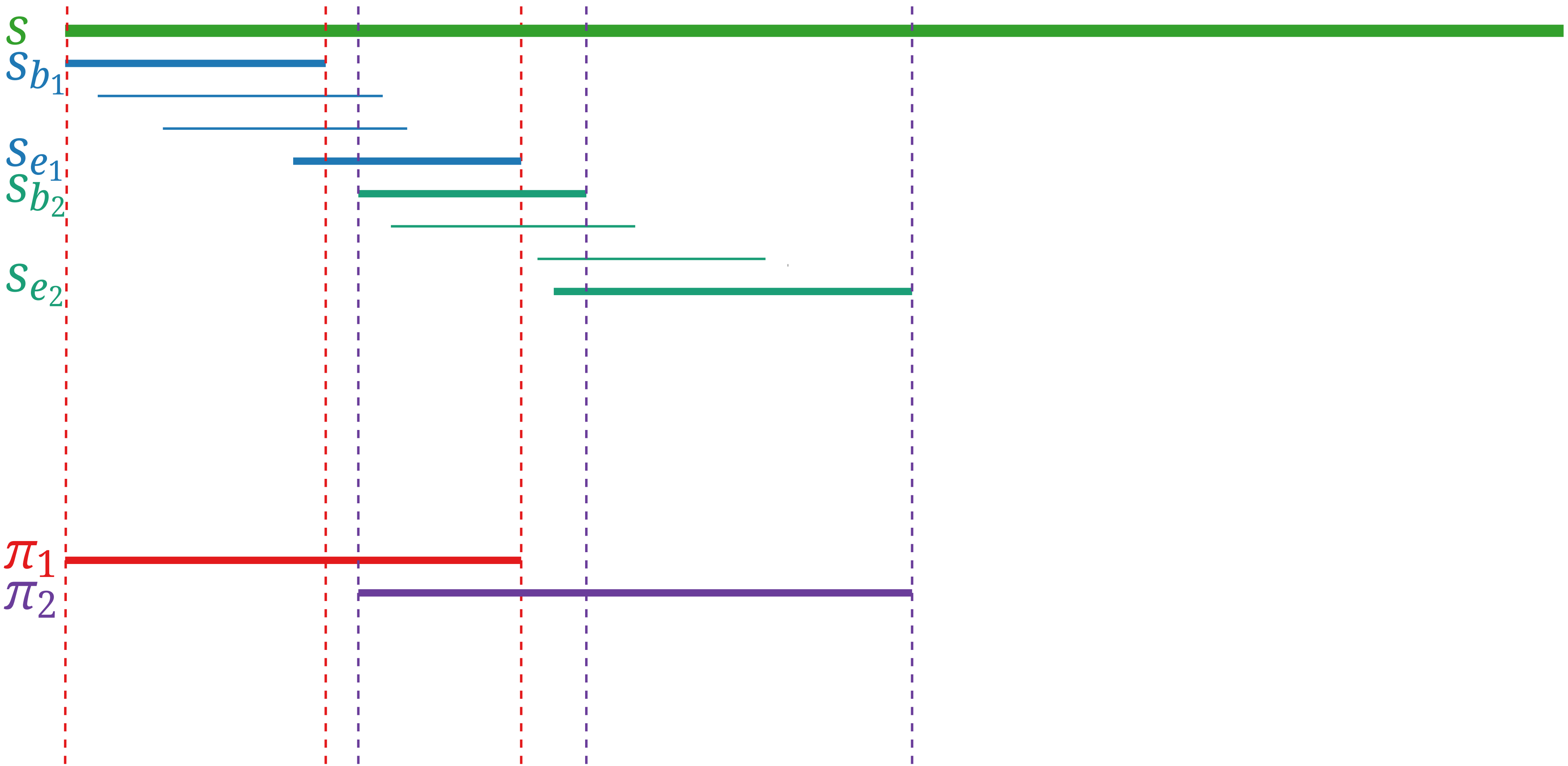
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SETCOVER

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

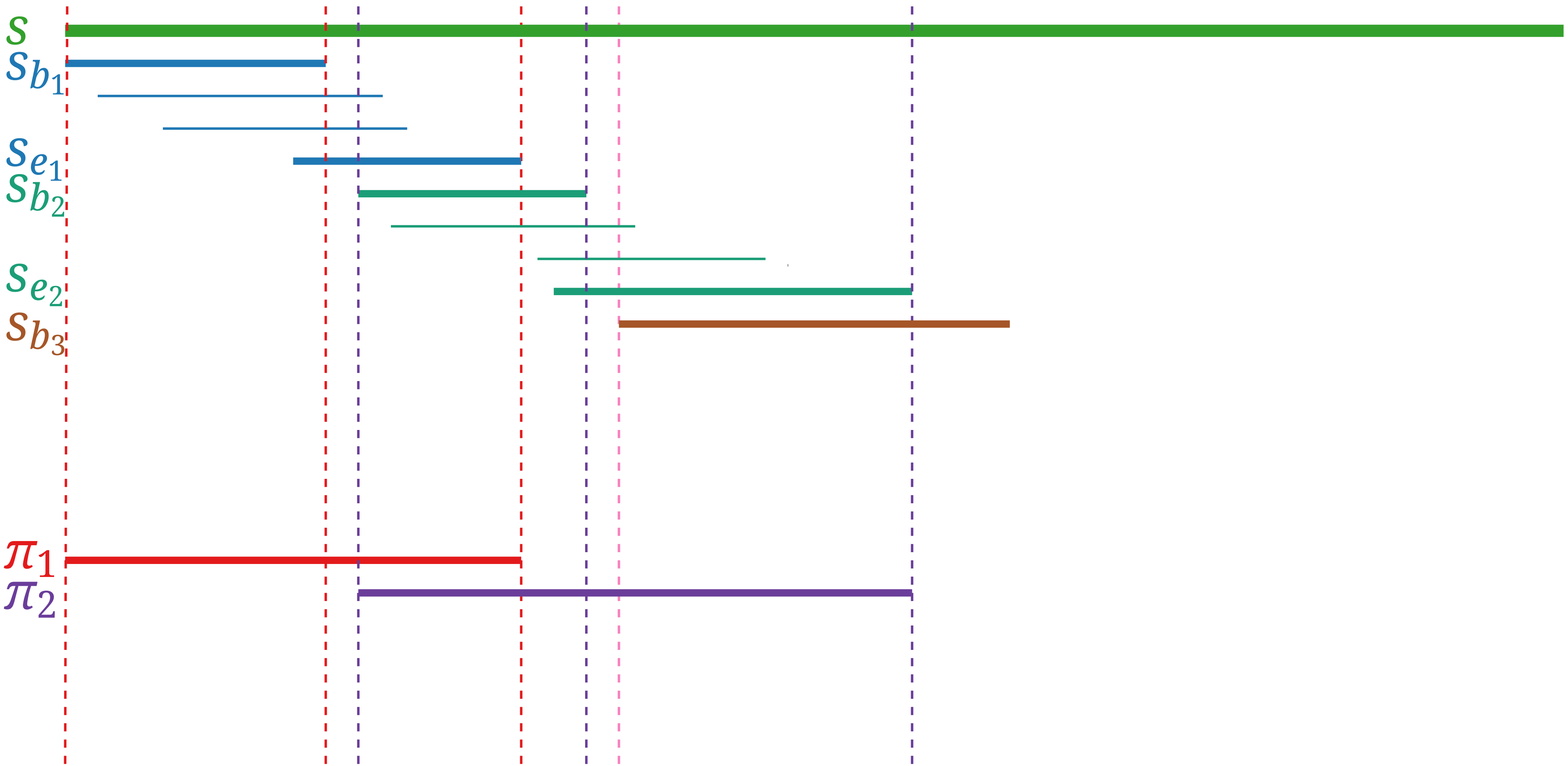
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

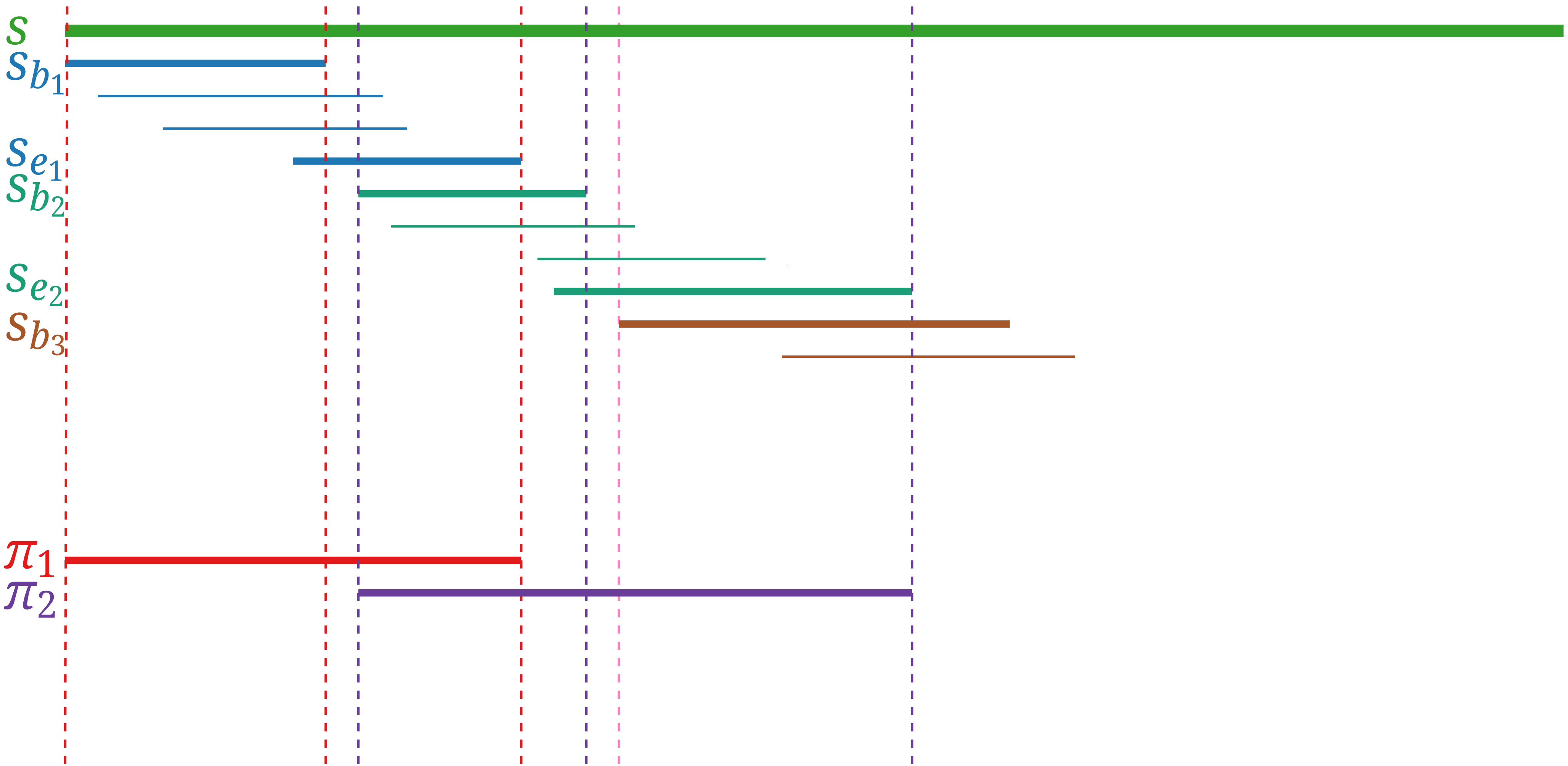
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

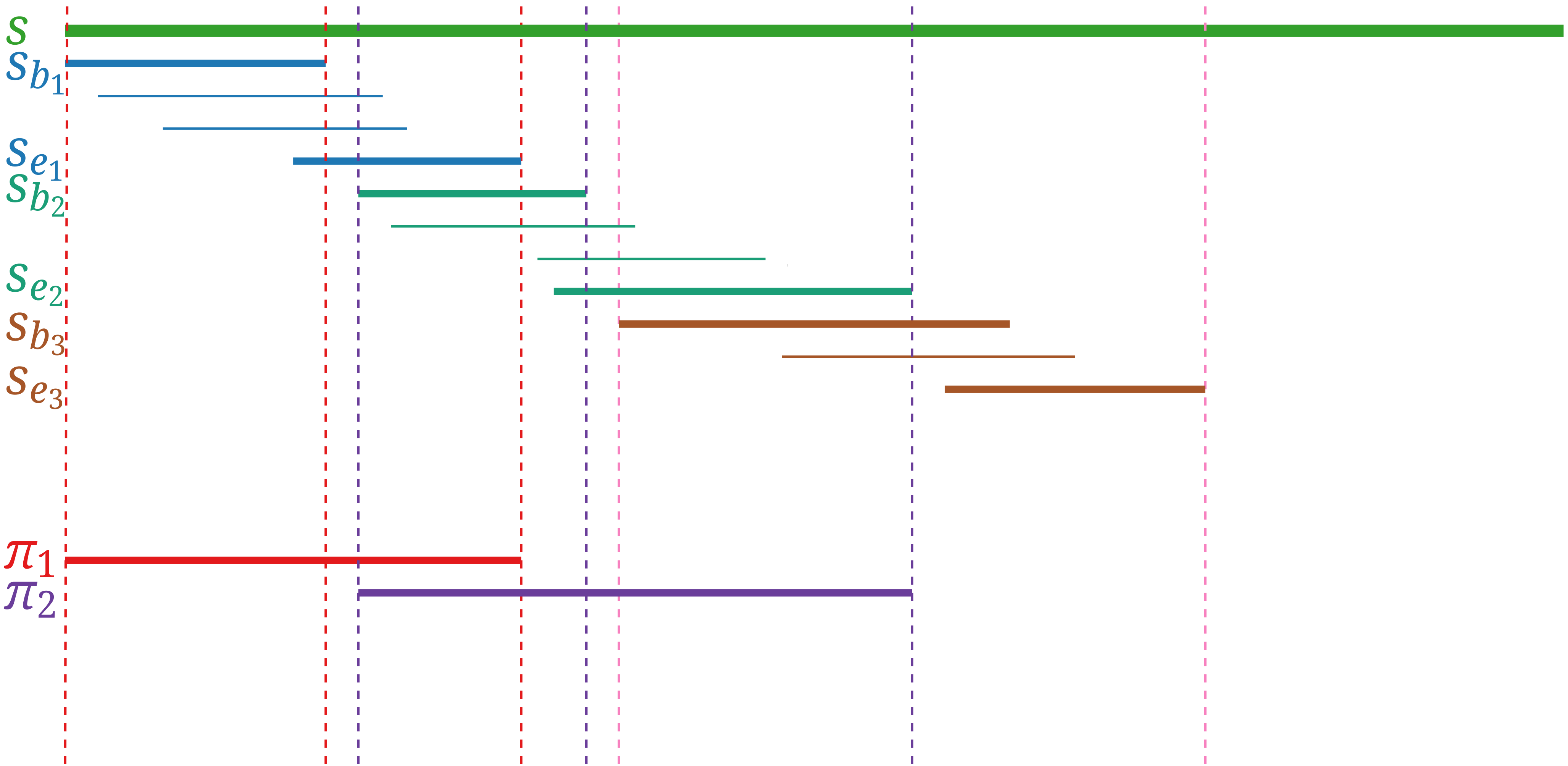
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SETCOVER

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

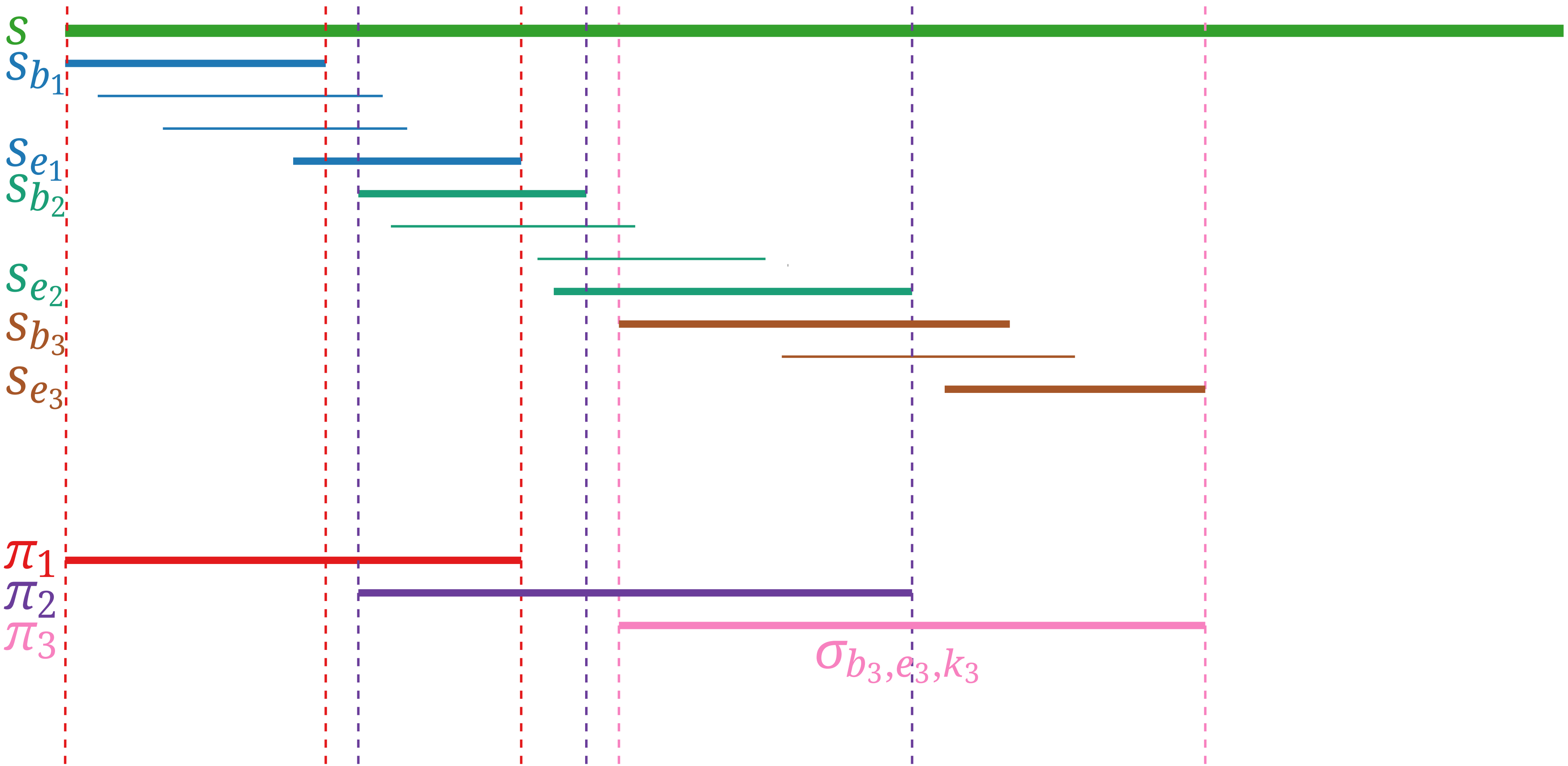
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

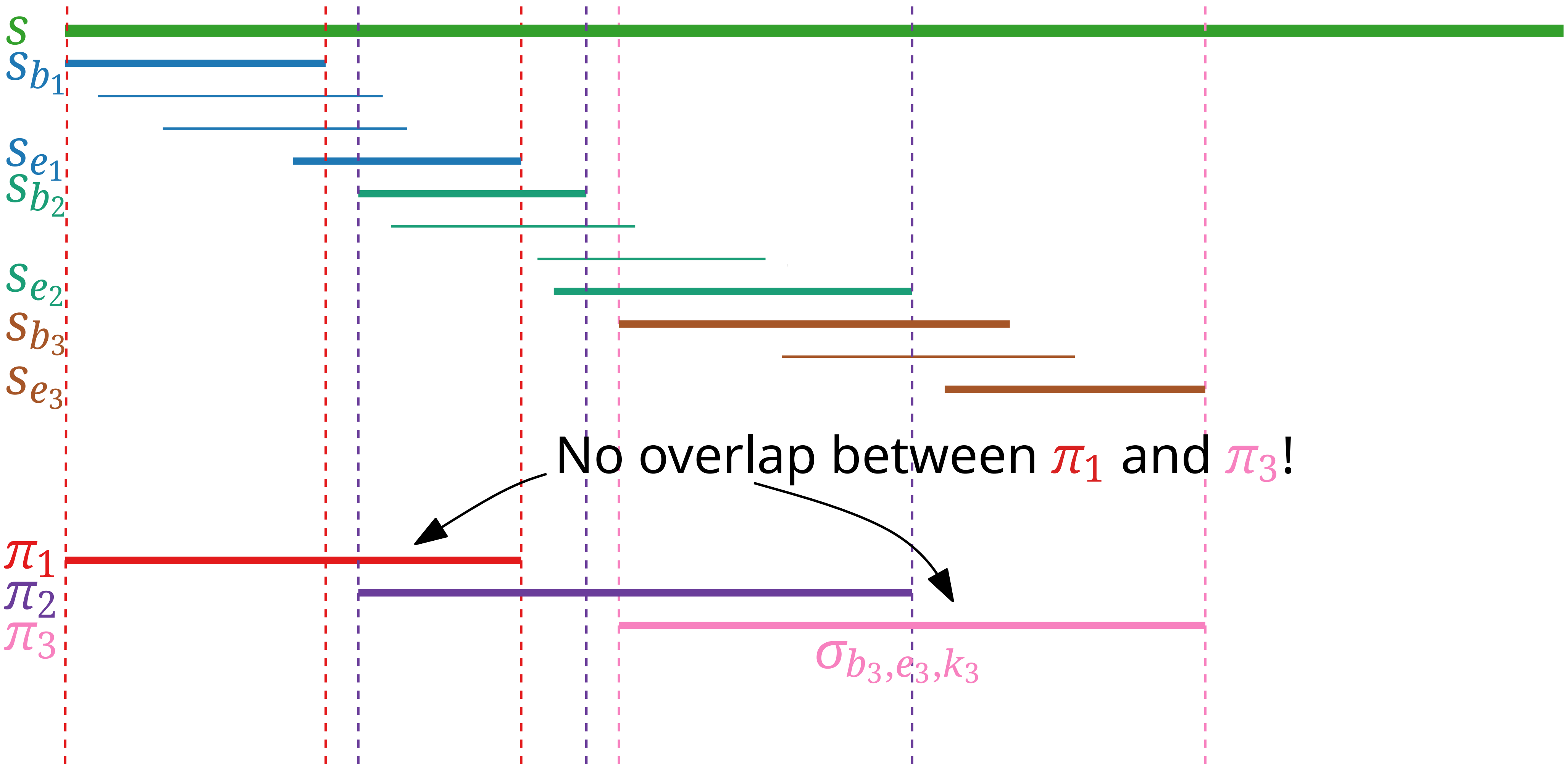
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

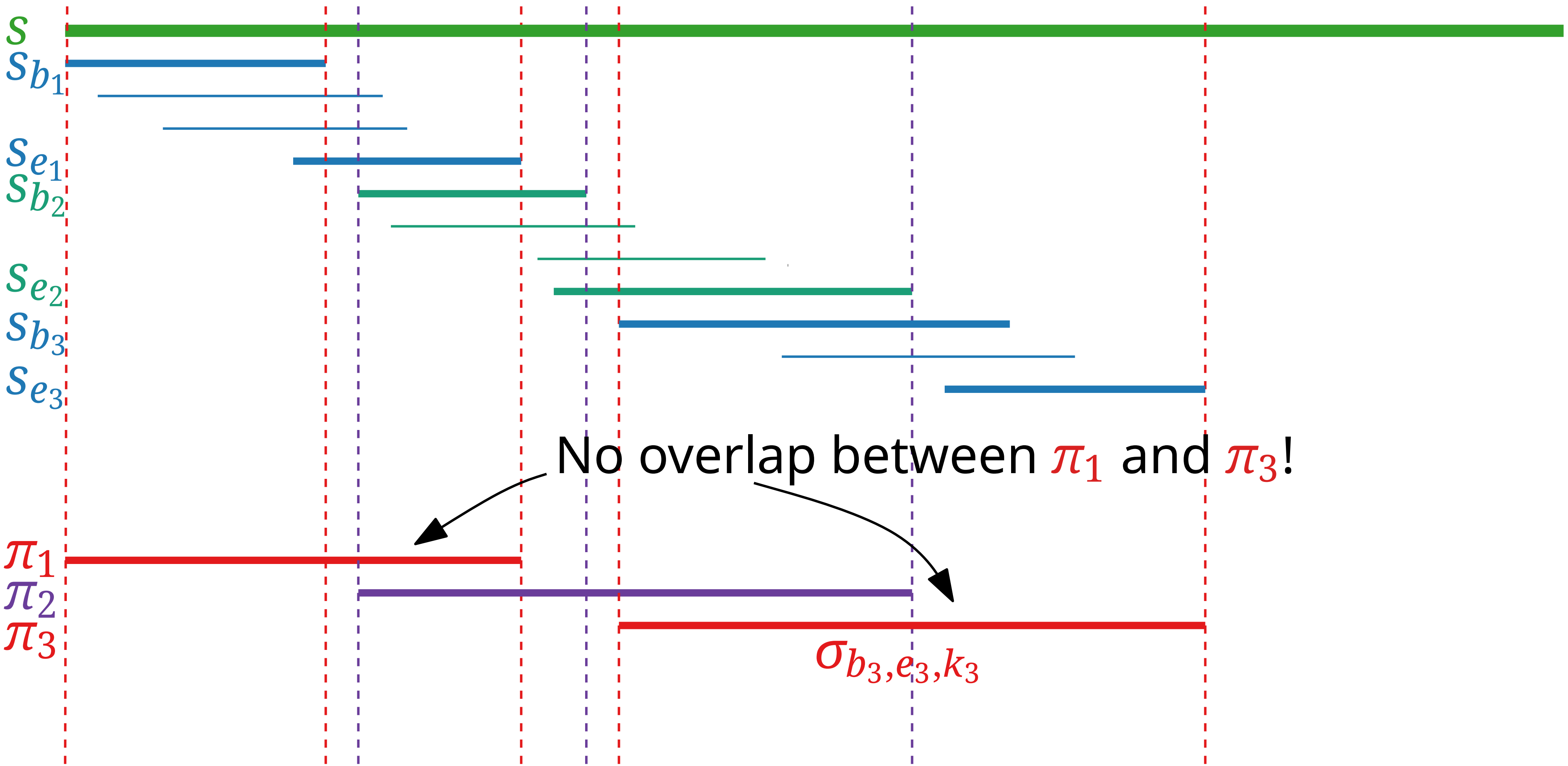
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

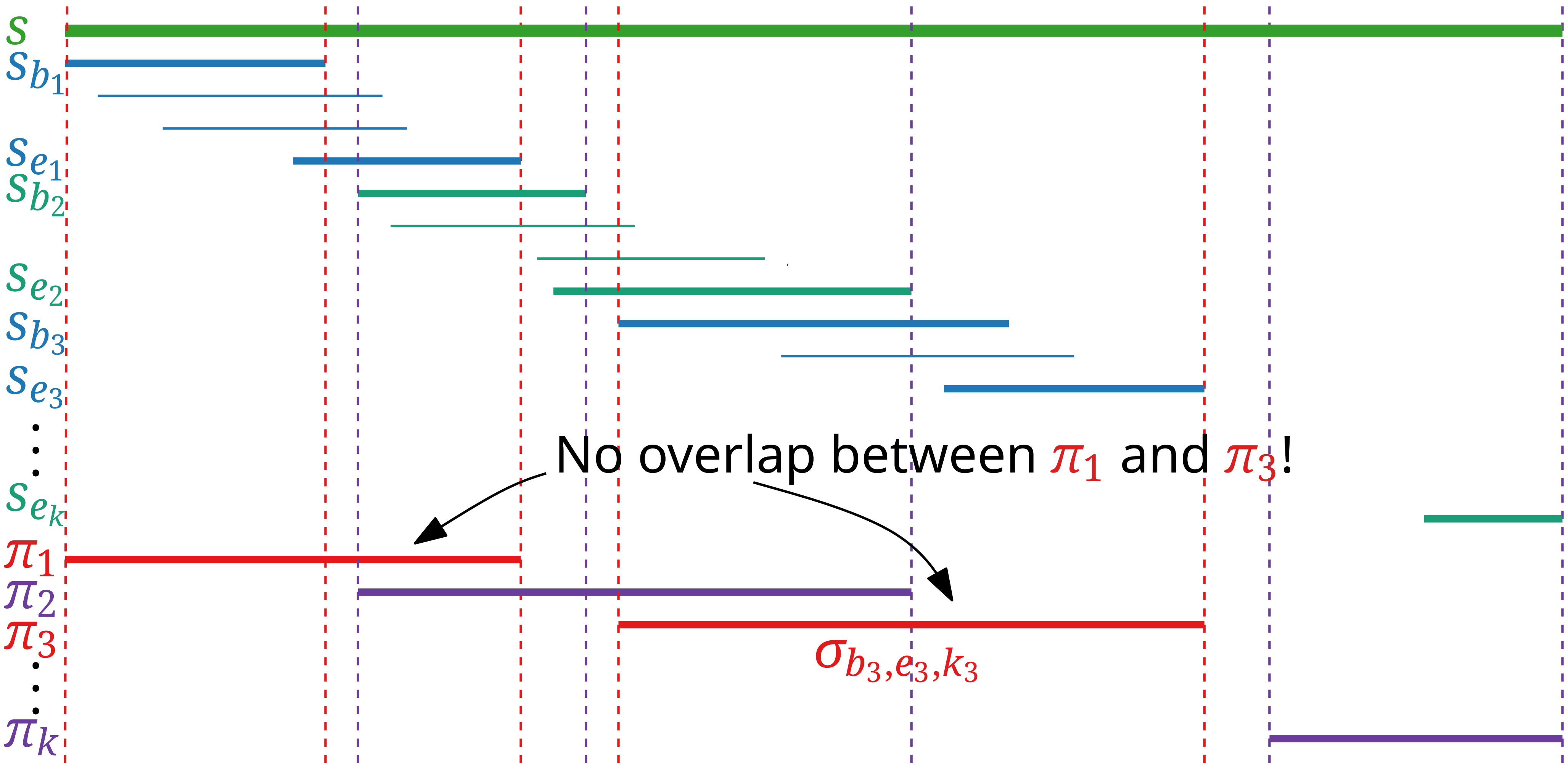
Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof. Consider an optimal superstring s.
Construct a set cover of cost $\leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.



Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof.

Each string $s_i \in U$ is a substring of some π_j.

Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof.

Each string $s_i \in U$ is a substring of some π_j.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is a solution for the SetCover instance with cost $\sum_i |\pi_i|$.

Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof.

Each string $s_i \in U$ is a substring of some π_j.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is a solution for the SetCover instance with cost $\sum_i |\pi_i|$.

Substrings $\pi_1, \ldots, \pi_k$ cover s, but π_j, π_{j+2} do not overlap.

Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof.

Each string $s_i \in U$ is a substring of some π_j.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is a solution for the SetCover instance with cost $\sum_i |\pi_i|$.

Substrings $\pi_1, \ldots, \pi_k$ cover s, but π_j, π_{j+2} do not overlap.

Hence each character in s lies in (at least one but) at most **two** (subsequent) substrings π_j and π_{j+1}.

Relating SSS and SetCover

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Proof.

Each string $s_i \in U$ is a substring of some π_j.

$\{S(\pi_1), \ldots, S(\pi_k)\}$ is a solution for the SetCover instance with cost $\sum_i |\pi_i|$.

Substrings $\pi_1, \ldots, \pi_k$ cover s, but π_j, π_{j+2} do not overlap.

Hence each character in s lies in (at least one but) at most **two** (subsequent) substrings π_j and π_{j+1}.

$\sum_i |\pi_i| \leq 2|s| = 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$

Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.

Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ with the algorithm GreedySetCover.

Algorithm for SSS

1. Construct SETCOVER instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ with the algorithm GreedySetCover.
3. Return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ with the algorithm GreedySetCover.
3. Return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Theorem. This algorithm is a factor-$2\mathcal{H}_n$ approximation algorithm for ShortestSuperString.

Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c \rangle$.
2. Compute a set cover $\{S(\pi_1), \ldots, S(\pi_k)\}$ with the algorithm GreedySetCover.
3. Return $\pi_1 \circ \cdots \circ \pi_k$ as the superstring.

Theorem. This algorithm is a factor-$2\mathcal{H}_n$ approximation algorithm for ShortestSuperString.

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Algorithm for SSS

1. Construct SetCover instance $\langle U, \mathcal{S}, c\rangle$.
2. Compute a set cover $\{\mathcal{S}(\pi_1), \dots, \mathcal{S}(\pi_k)\}$ with the algorithm GreedySetCover.
3. Return $\pi_1 \circ \dots \circ \pi_k$ as the superstring.

Theorem. This algorithm is a factor-$2\mathcal{H}_n$ approximation algorithm for ShortestSuperString.

Lemma. $\mathrm{OPT}_{\mathrm{SC}} \leq 2 \cdot \mathrm{OPT}_{\mathrm{SSS}}$.

Theorem. GreedySetCover is a factor-$\mathcal{H}_k$ approximation algorithm for SetCover, where k is the cardinality of the largest set in $\mathcal{S}$ and
$\mathcal{H}_k := 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \leq 1 + \ln k$.

Can we do better?

Can we do better?

- The best known approximation factor for SHORTESTSUPERSTRING is $\frac{71}{30} \approx 2.367$.

Can we do better?

- The best known approximation factor for ShortestSuperString is $\frac{71}{30} \approx 2.367$.
- ShortestSuperString cannot be approximated within factor $\frac{333}{332} \approx 1.003$ (unless $\mathrm{P} = \mathrm{NP}$).

Summary

set cover: greedy algorithm is $O(\log k)$ approximation where k is the size of the largest set

Summary

set cover: greedy algorithm is $O(\log k)$ approximation where k is the size of the largest set

vertex cover: special case of set cover. Therefore, same approximation ratio, which is worse than 2-approximation that we have seen for min-cardinality vertex cover

Summary

set cover: greedy algorithm is $O(\log k)$ approximation where k is the size of the largest set

vertex cover: special case of set cover. Therefore, same approximation ratio, which is worse than 2-approximation that we have seen for min-cardinality vertex cover

weighted vertex cover: 2-approximation using local ratio (aka layering)

Summary

set cover: greedy algorithm is $O(\log k)$ approximation where k is the size of the largest set

vertex cover: special case of set cover. Therefore, same approximation ratio, which is worse than 2-approximation that we have seen for min-cardinality vertex cover

weighted vertex cover: 2-approximation using local ratio (aka layering)

shortest superstring: can be approximated using set cover with extra factor 2: approximation algorithm with factor $2\mathcal{H}_n = O(\log n)$

Summary

set cover: greedy algorithm is $O(\log k)$ approximation where k is the size of the largest set

vertex cover: special case of set cover. Therefore, same approximation ratio, which is worse than 2-approximation that we have seen for min-cardinality vertex cover

weighted vertex cover: 2-approximation using local ratio (aka layering)

shortest superstring: can be approximated using set cover with extra factor 2: approximation algorithm with factor $2\mathcal{H}_n = O(\log n)$

shortest superstring: better approximation algorithms exist