Algorithms

Lecture Topic: Approximation Algorithms (Part 1)

Anxiao (Andrew) Jiang

Roadmap of this lecture:

1. Define "Approximation Algorithm".

2. Understand approximation algorithms by solving the "Vertex Cover Problem".

   2.1 An approximation algorithm for "Vertex Cover Problem".

   2.2 Analyze the approximation ratio of the algorithm.

3. Understand approximation algorithms by solving the "Traveling Salesman Problem (TSP)".

   3.1 An approximation algorithm for TSP with the triangle inequality.

   3.2 Analyze the approximation ratio of the algorithm.

Approximation Algorithms

Why do we need Approximation Algorithms?

Approximation algorithms are “easier” than exact algorithms, since it requires only approximate solutions, not optimal solutions.

How to analyze approximation algorithms?

Approximation Algorithms

Consider an optimization problem.

Let C^* be the cost of an optimal solution.

Let C be the cost of the solution found by our algorithm.

(For simplicity, assume cost > 0.)

Approximation Algorithms

maximization

Consider an optimization problem.

Let C^* be the cost of an optimal solution.

Let C be the cost of the solution found by our algorithm.

(For simplicity, assume cost > 0.)

Then $C^* \geq C$, $\dfrac{C^*}{C} \geq 1.$

If $\dfrac{C^*}{C} \leq \rho$ for all possible instances, then our algorithm is called a ρ-approximation algorithm.

We say our algorithm has an approximation ratio of ρ .

Approximation Algorithms

minimization

Consider an optimization problem.

Let C^* be the cost of an optimal solution.

Let C be the cost of the solution found by our algorithm.

(For simplicity, assume cost > 0.)

Then $C^* \leq C,$ $\frac{C}{C^*} \geq 1.$

If $\frac{C}{C^*} \leq \rho$ for all possible instances, then our algorithm is called a ρ-approximation algorithm.

We say our algorithm has an approximation ratio of ρ .

Quiz questions:

1. What is an "Approximation Algorithm"?

2. What is "approximation ratio"?

Roadmap of this lecture:

1. Define "Approximation Algorithm".

2. Understand approximation algorithms by solving the "Vertex Cover Problem".

    2.1 An approximation algorithm for "Vertex Cover Problem".

    2.2 Analyze the approximation ratio of the algorithm.

3. Understand approximation algorithms by solving the "Traveling Salesman Problem (TSP)".

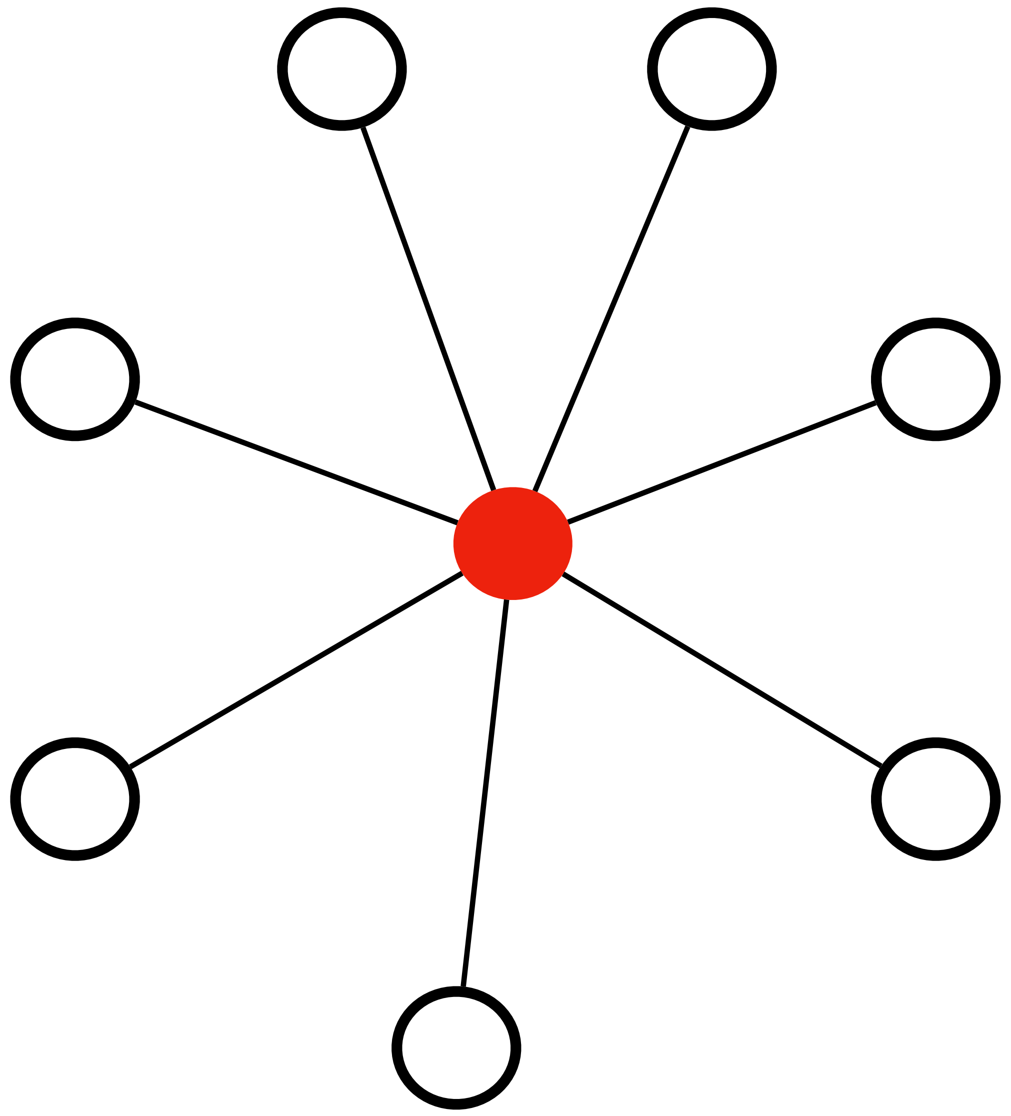
    3.1 An approximation algorithm for TSP with the triangle inequality.

    3.2 Analyze the approximation ratio of the algorithm.

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.



Vertex Cover Problem

We will show a 2-approximation algorithm.

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Algorithm:

1. $S = \emptyset$ — S is the vertex cover.
2. $E' = E$ — E' are the uncovered edges.
3. while $E' \neq \emptyset$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

We will show a 2-approximation algorithm.

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Algorithm:

1. $S = \emptyset$ S is the vertex cover.

2. $E' = E$ E' are the uncovered edges.

3. while $E' \neq \emptyset$

4. let (u, v) be any edge in E'

5. $S \leftarrow S \cup \{u, v\}$

6. Remove all the edges that have either u or v as endpoints from E'

7. Return S

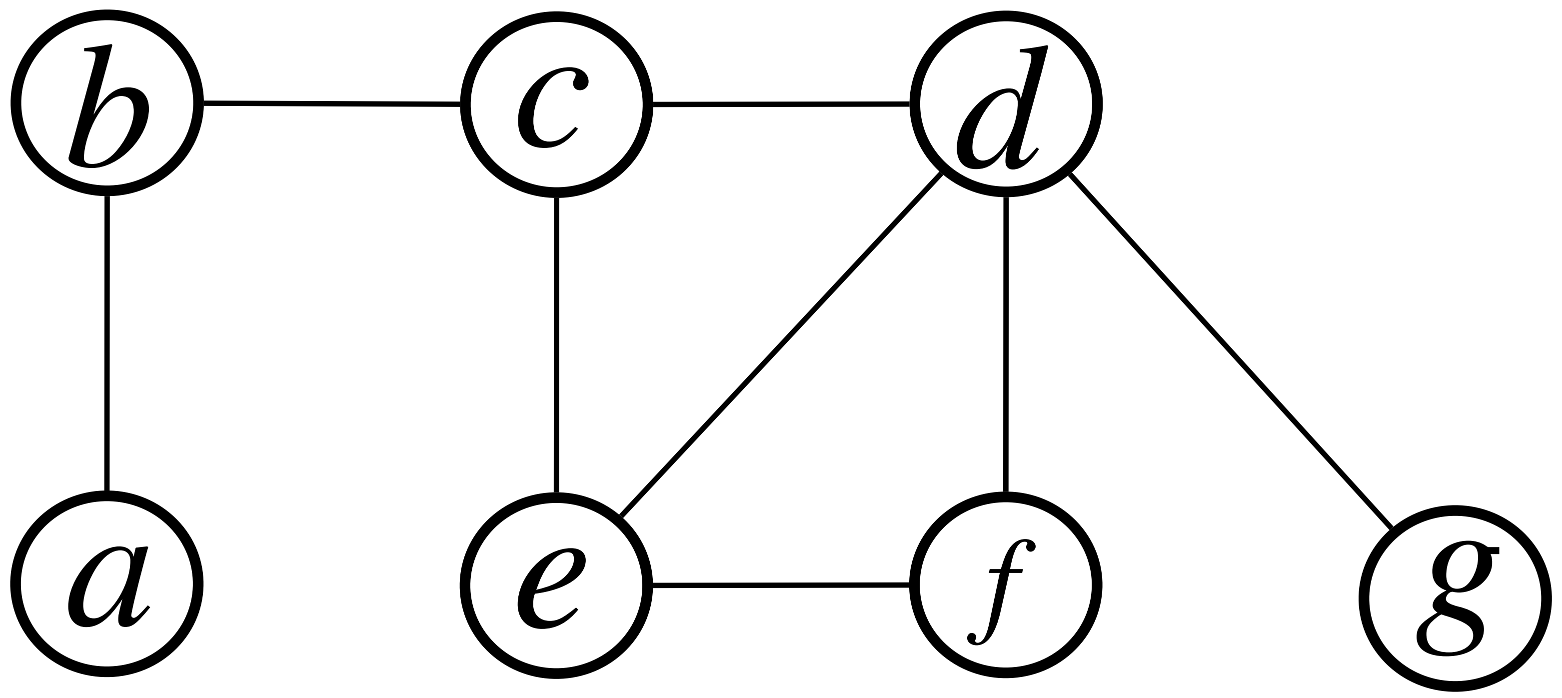
This step seems strange.
Either u or v is enough for covering the edge (u,v).
But we choose both u and v.

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Example:



Algorithm:

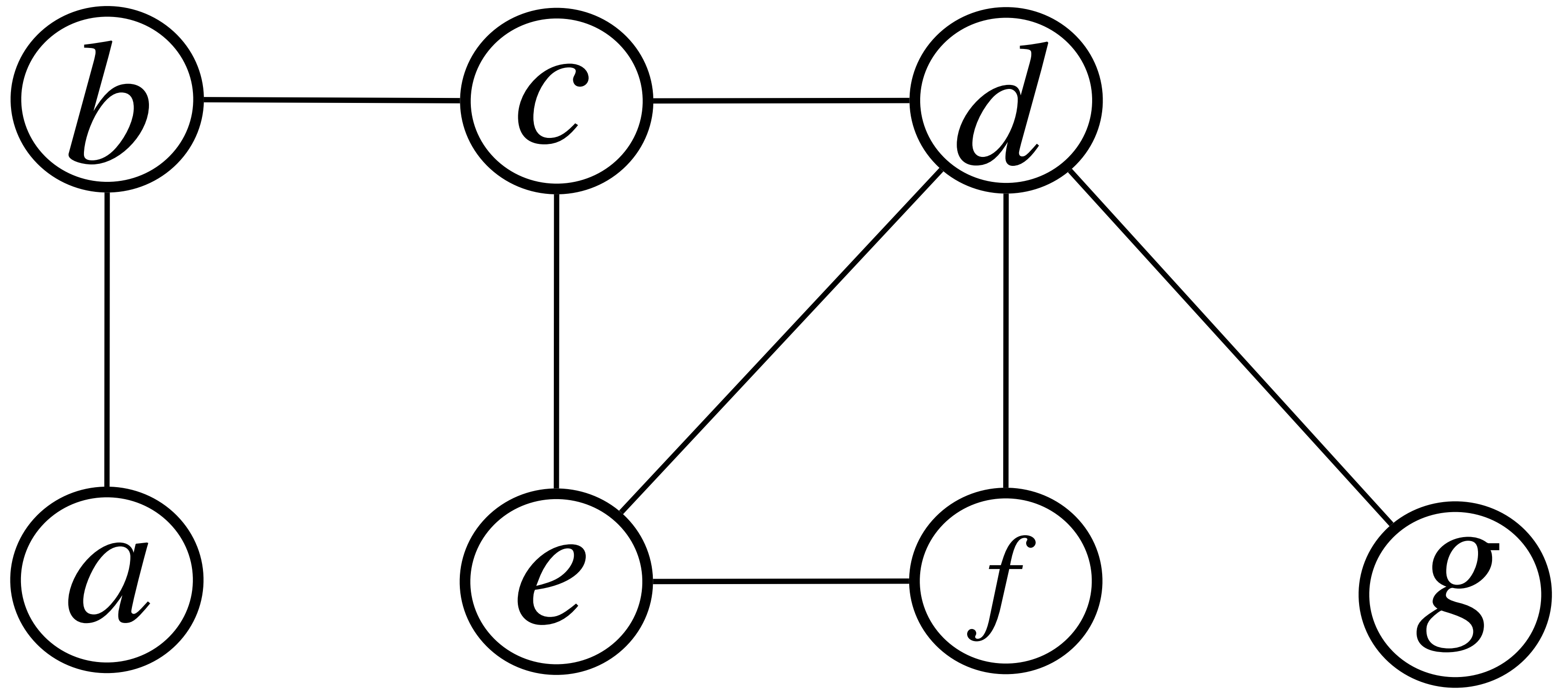
1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Example:



$S = \{ \quad \}$

Algorithm:

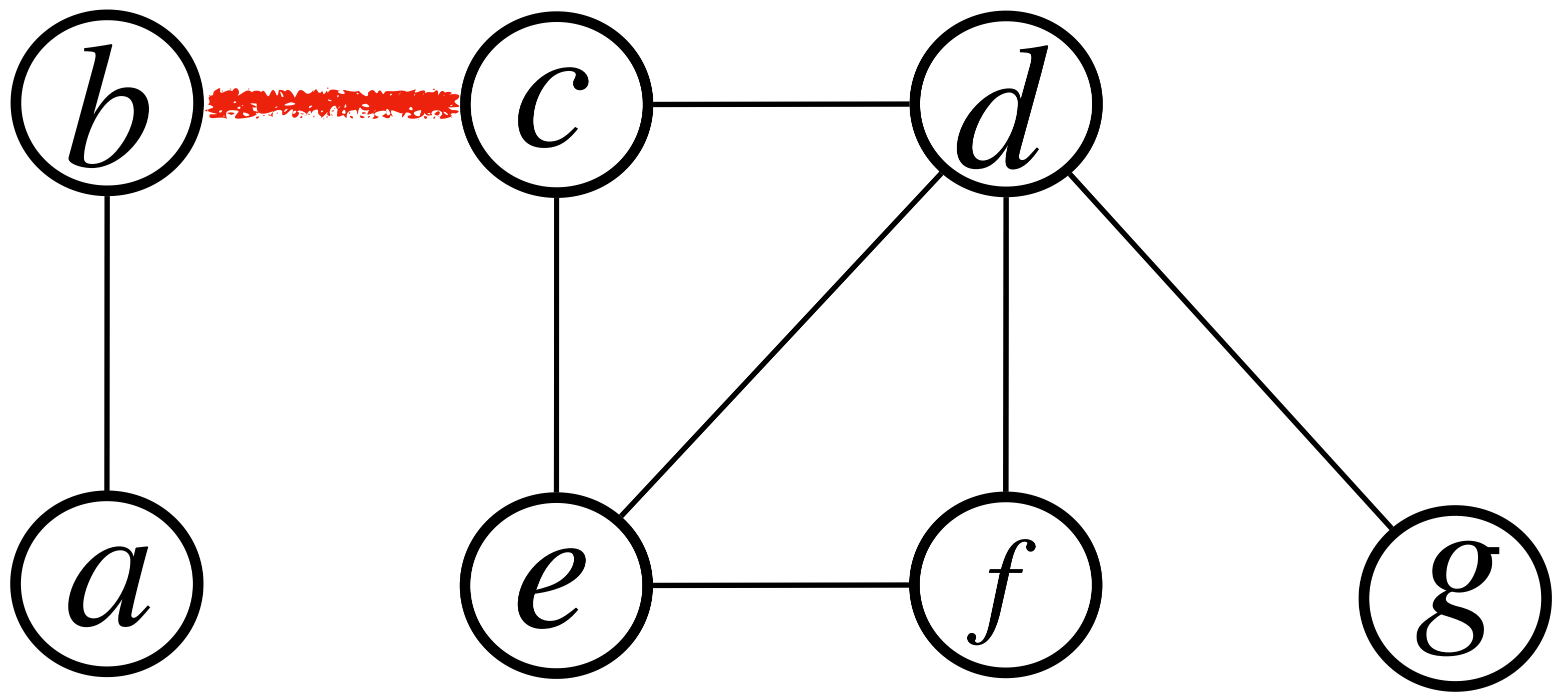
1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Example:



$S = \{ \quad \}$

Algorithm:

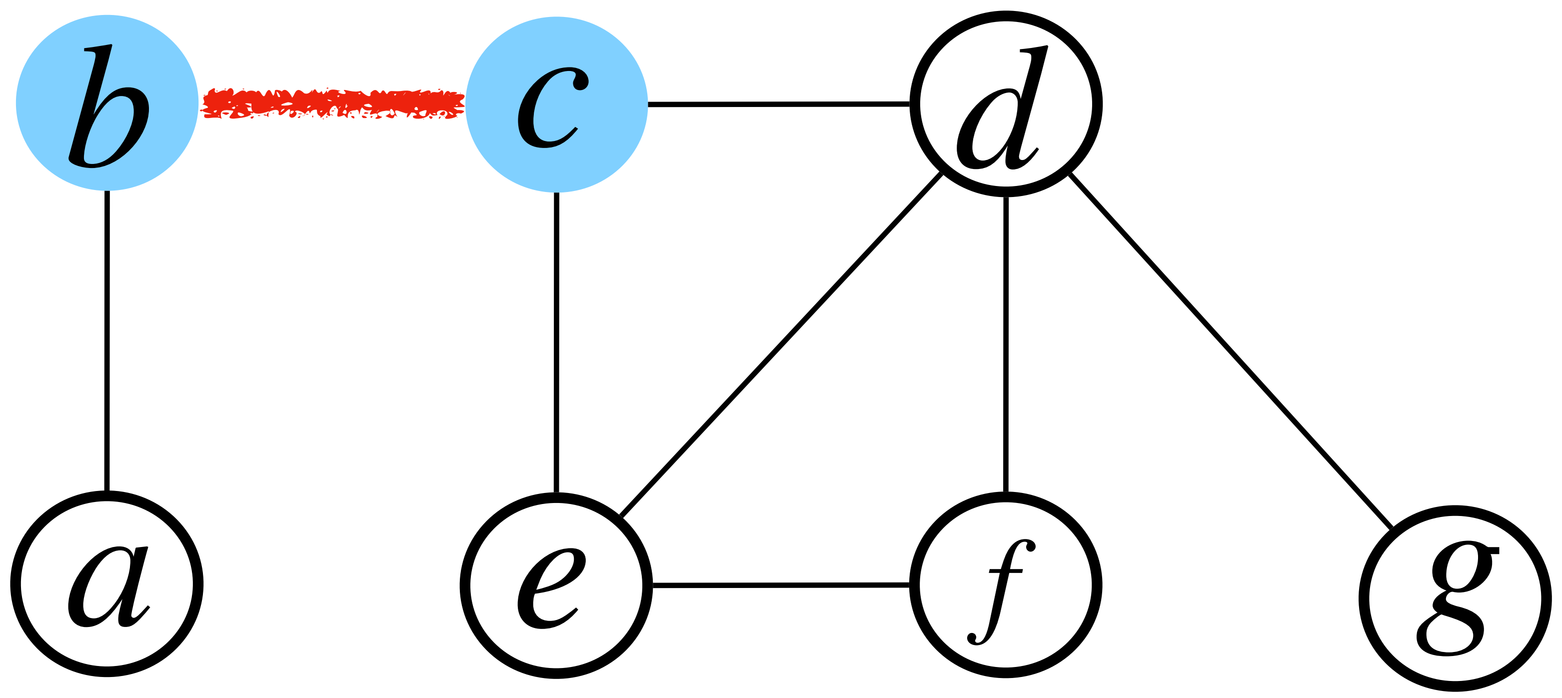
1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Example:



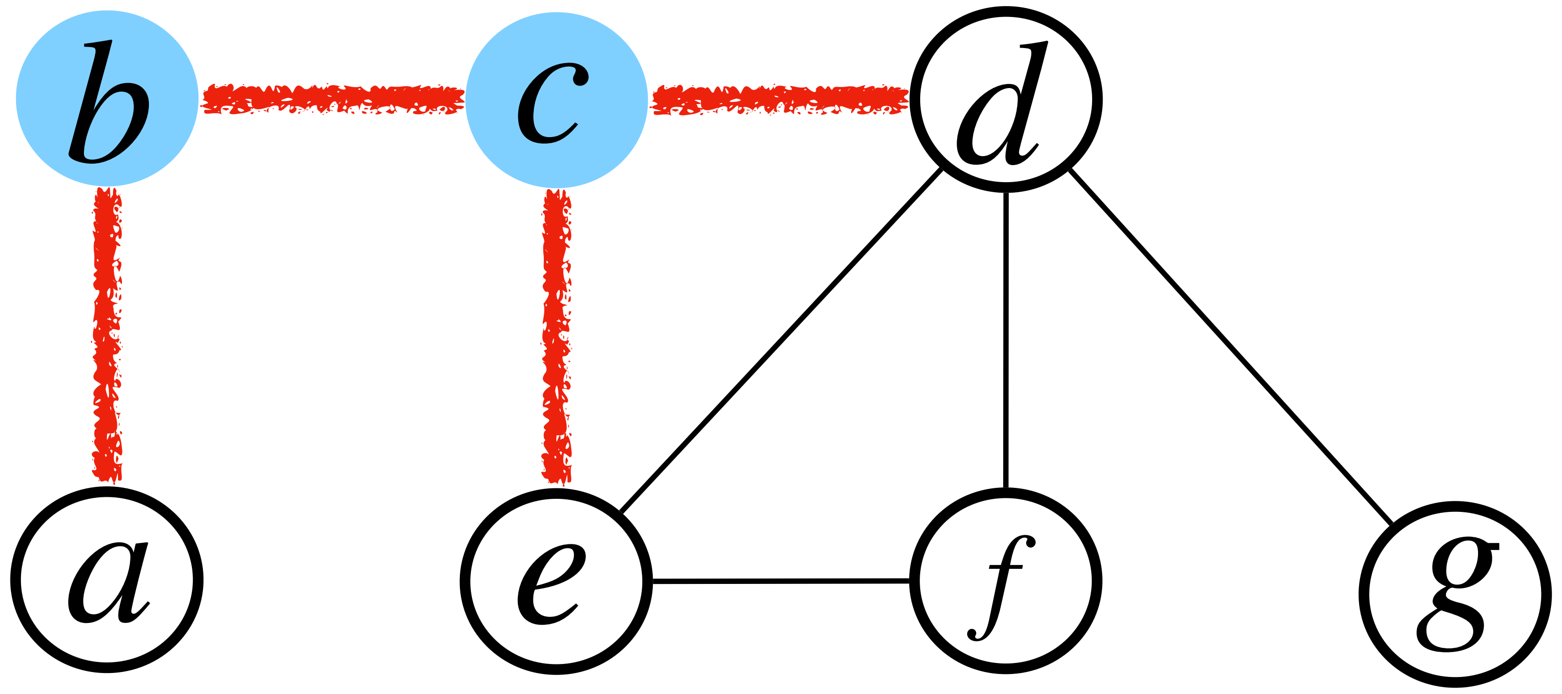
$S = \{b, c\}$

Algorithm:

1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem
Input: An undirected graph G=(V,E).
Output: A vertex cover of G of minimum size.

Example:



$S = \{b, c\}$

Algorithm:

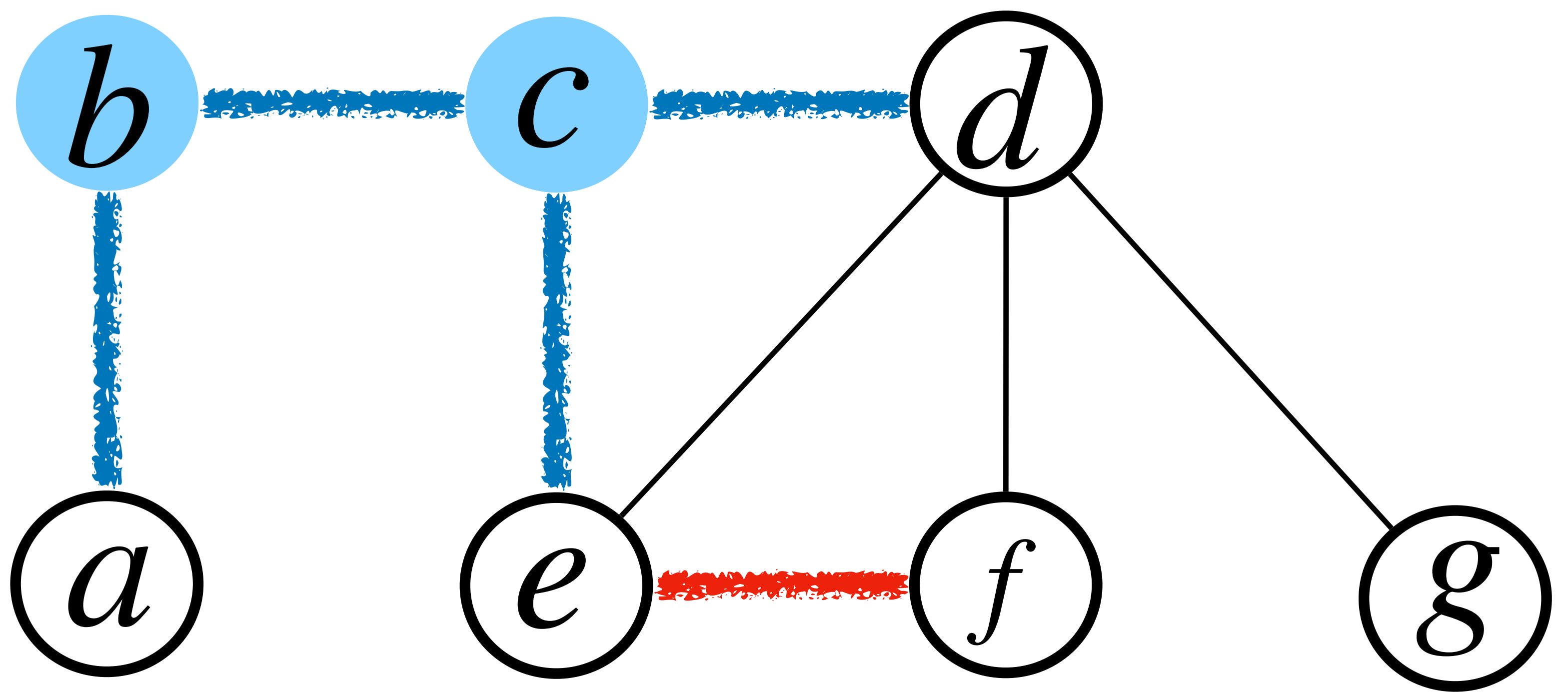
1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Example:



$S = \{b, c\}$

Algorithm:

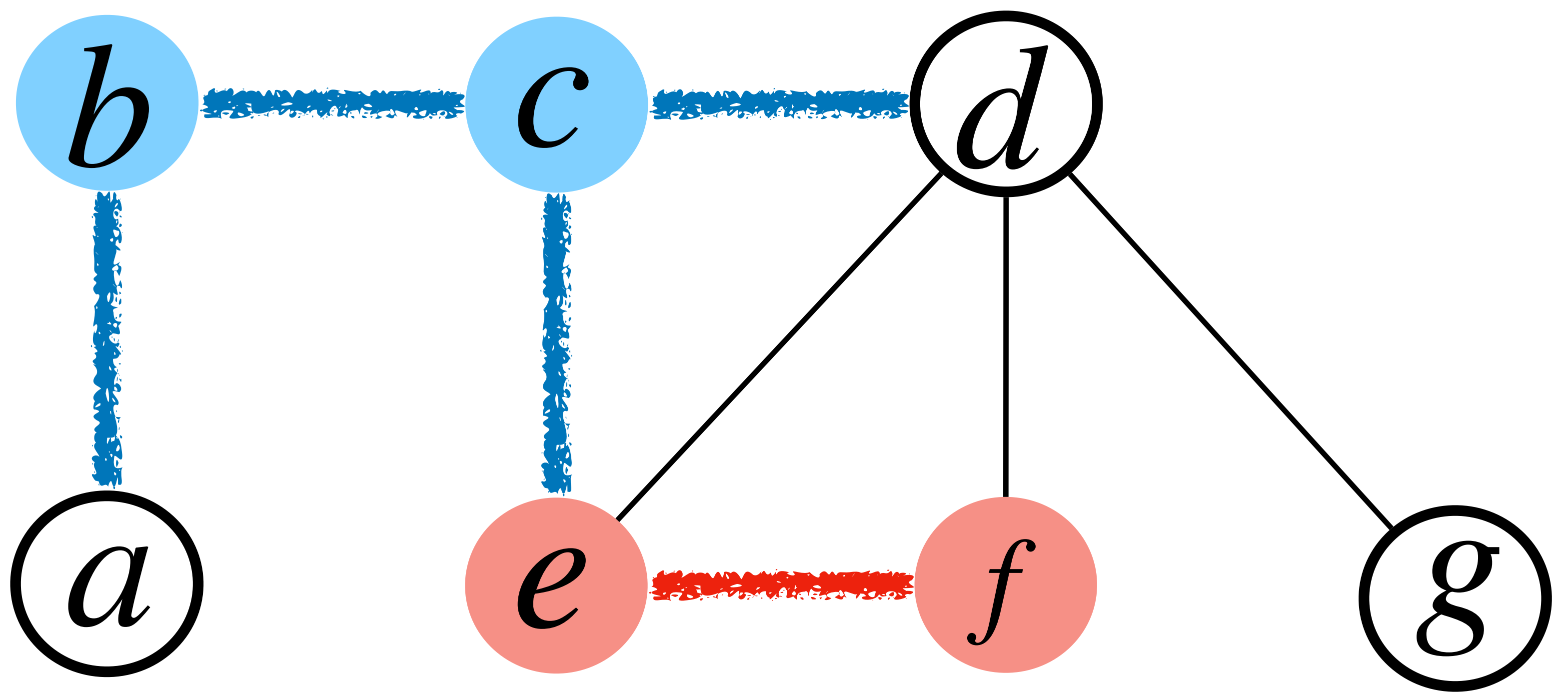
1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Example:



$$S = \{b, c, e, f\}$$

Algorithm:

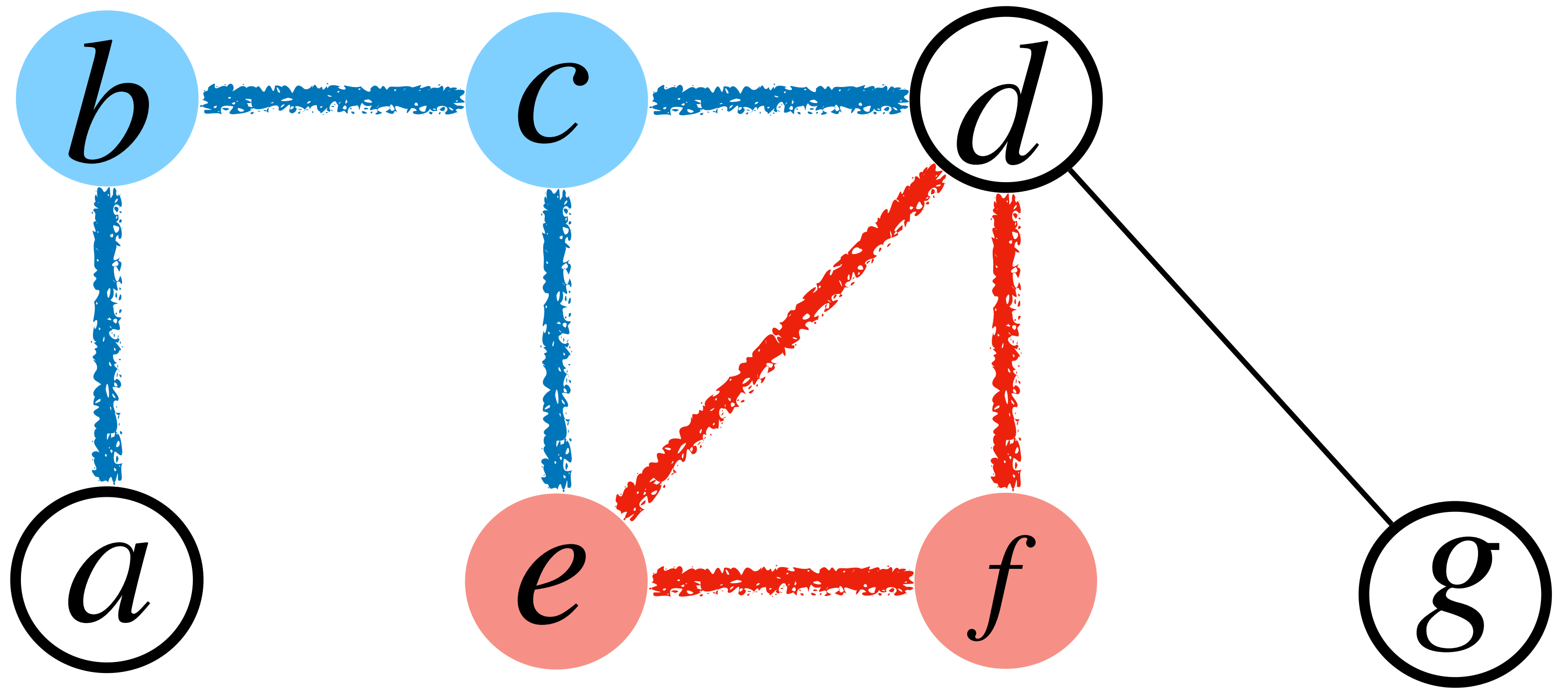
1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Example:



$$S = \{b, c, e, f\}$$

Algorithm:

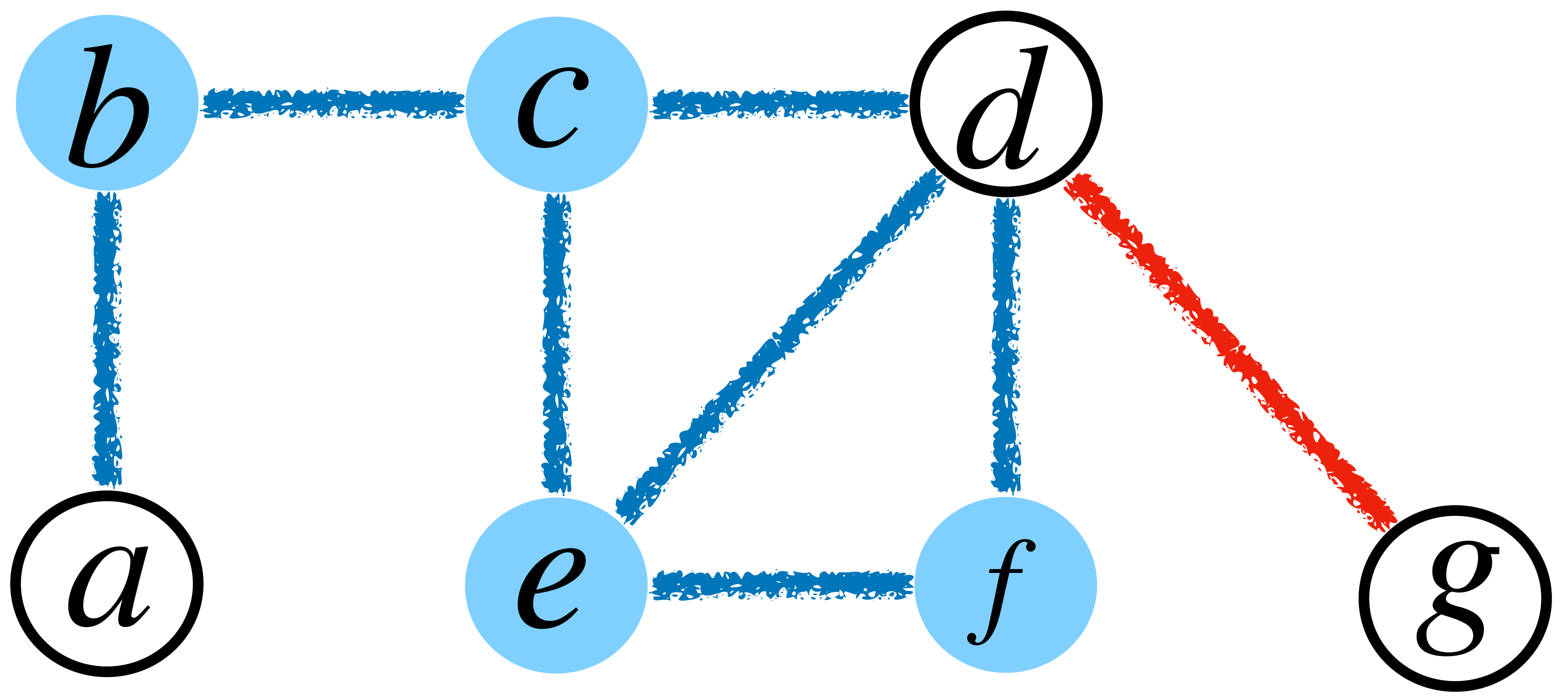
1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

Example:



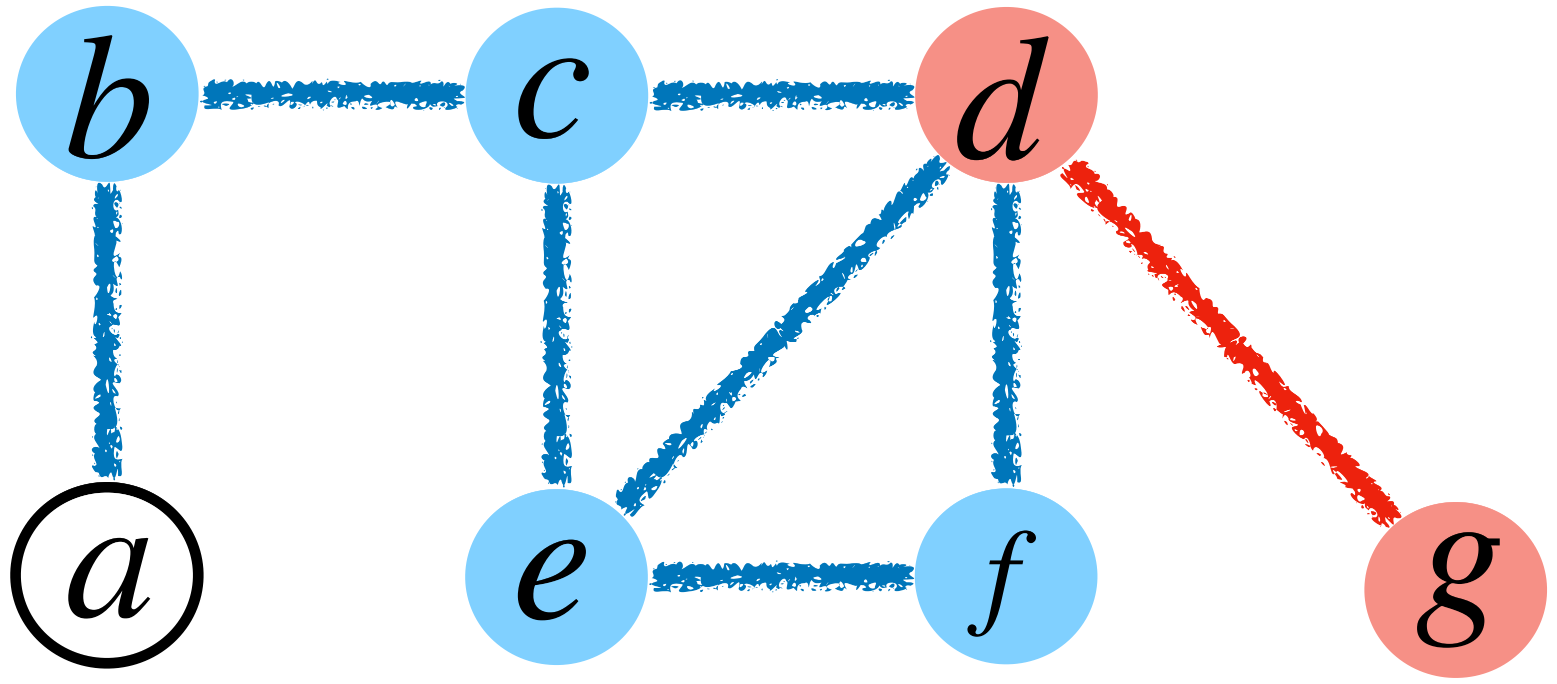
$$S = \{b, c, e, f\}$$

Algorithm:

1. $S = \varnothing$ — S is the vertex cover.
2. $E' = E$ — E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem
Input: An undirected graph G=(V,E).
Output: A vertex cover of G of minimum size.

Example:



$$S = \{b, c, e, f, d, g\}$$

Algorithm:

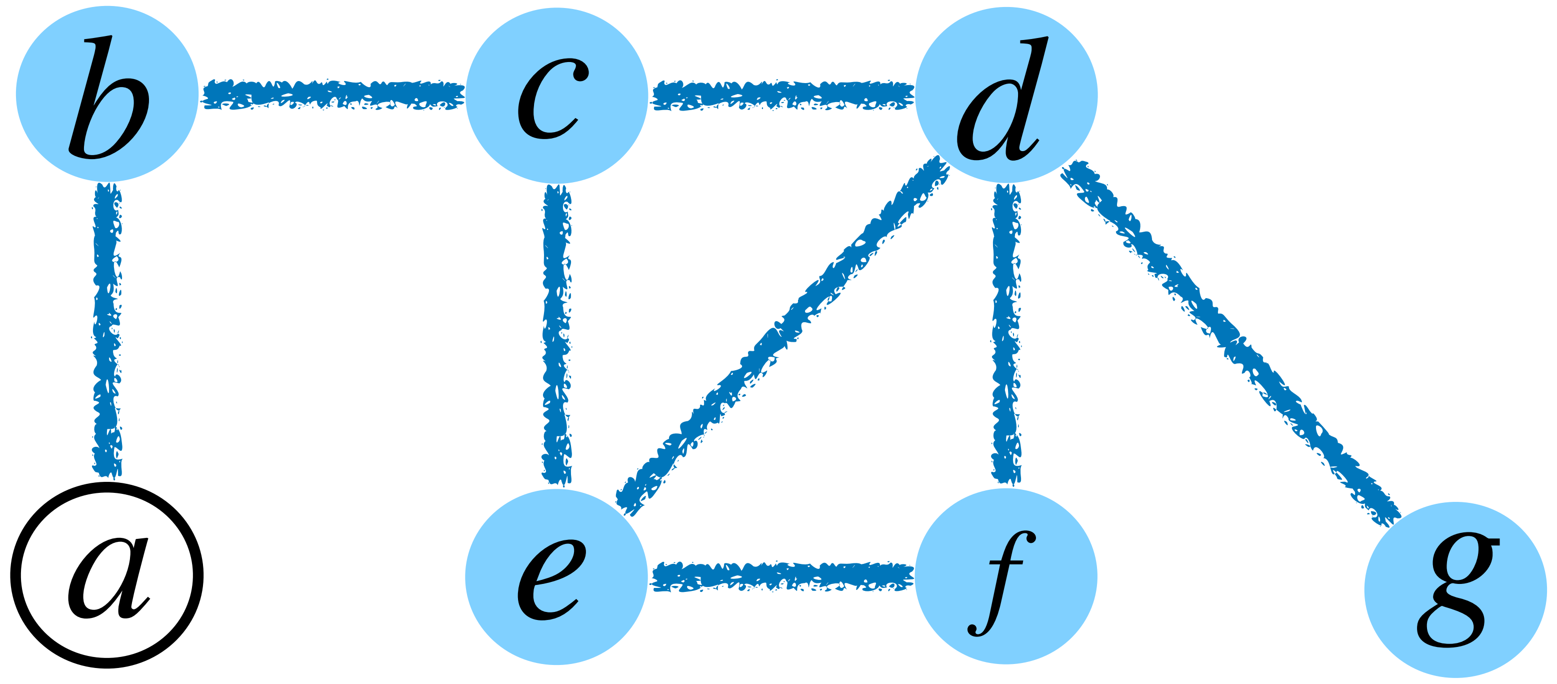
1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem

Input: An undirected graph G=(V,E).

Output: A vertex cover of G of minimum size.

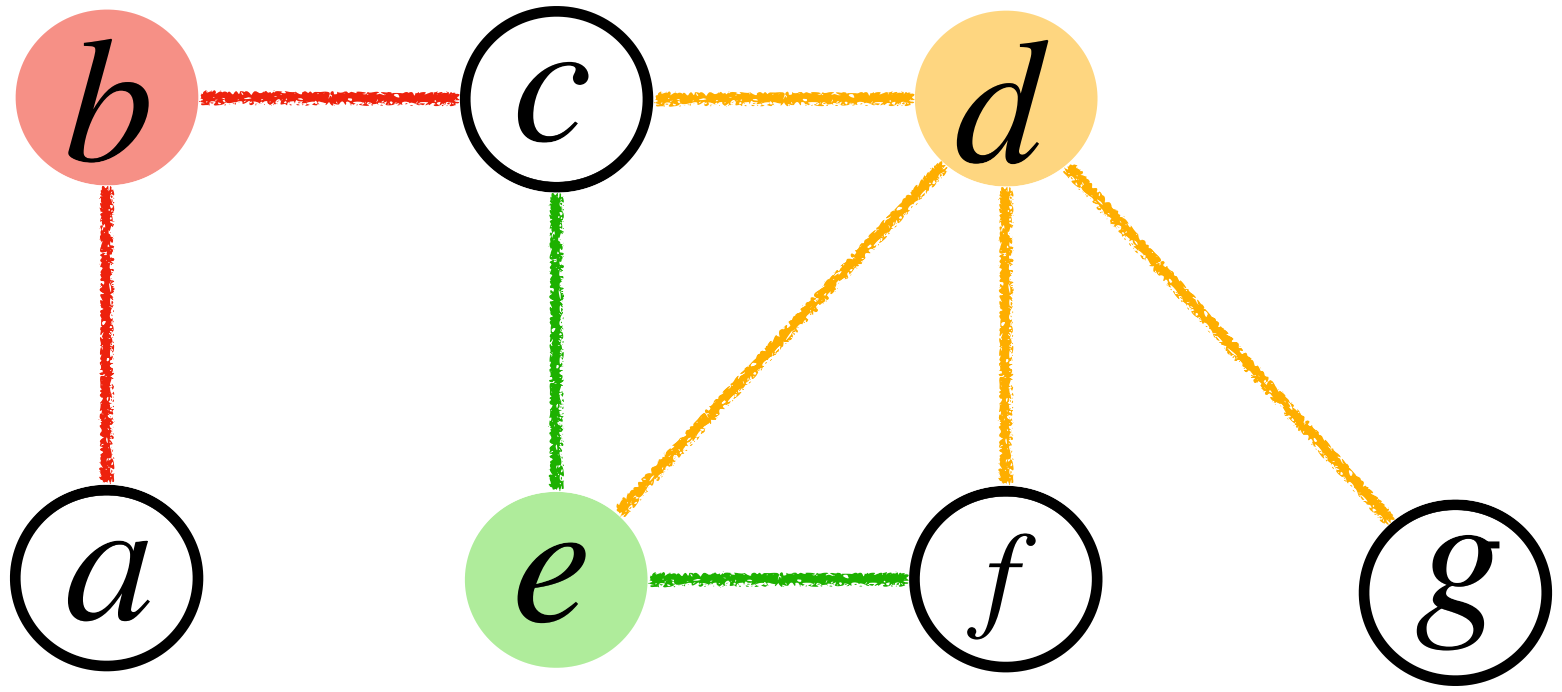
Example:



$$S = \{b, c, e, f, d, g\}$$

$$C = |S| = 6$$

$$C^* = 3$$



$$\frac{C}{C^*} = 2$$

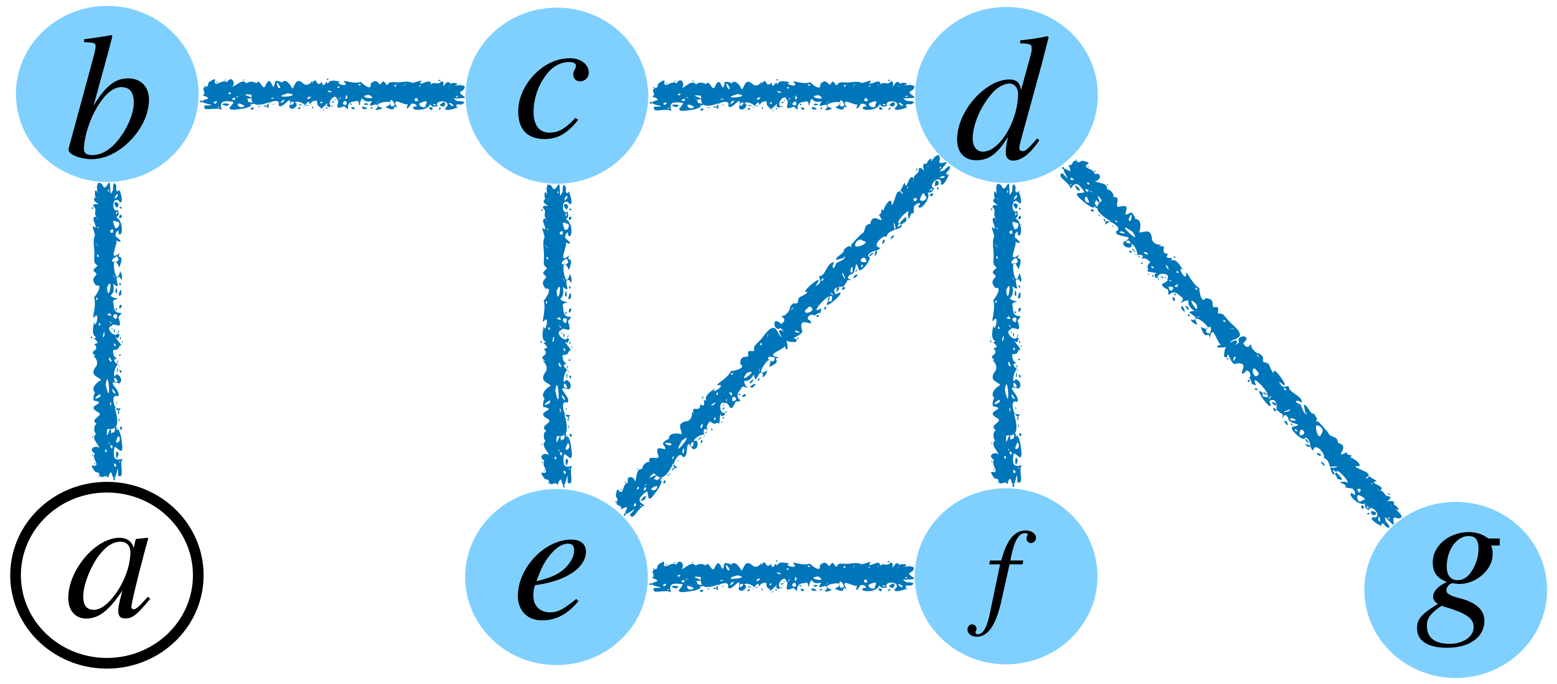
Algorithm:

1. $S = \varnothing$ S is the vertex cover.
2. $E' = E$ E' are the uncovered edges.
3. while $E' \neq \varnothing$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Vertex Cover Problem
Input: An undirected graph G=(V,E).
Output: A vertex cover of G of minimum size.

We need to show the approximation ratio is at most 2 for all instances.

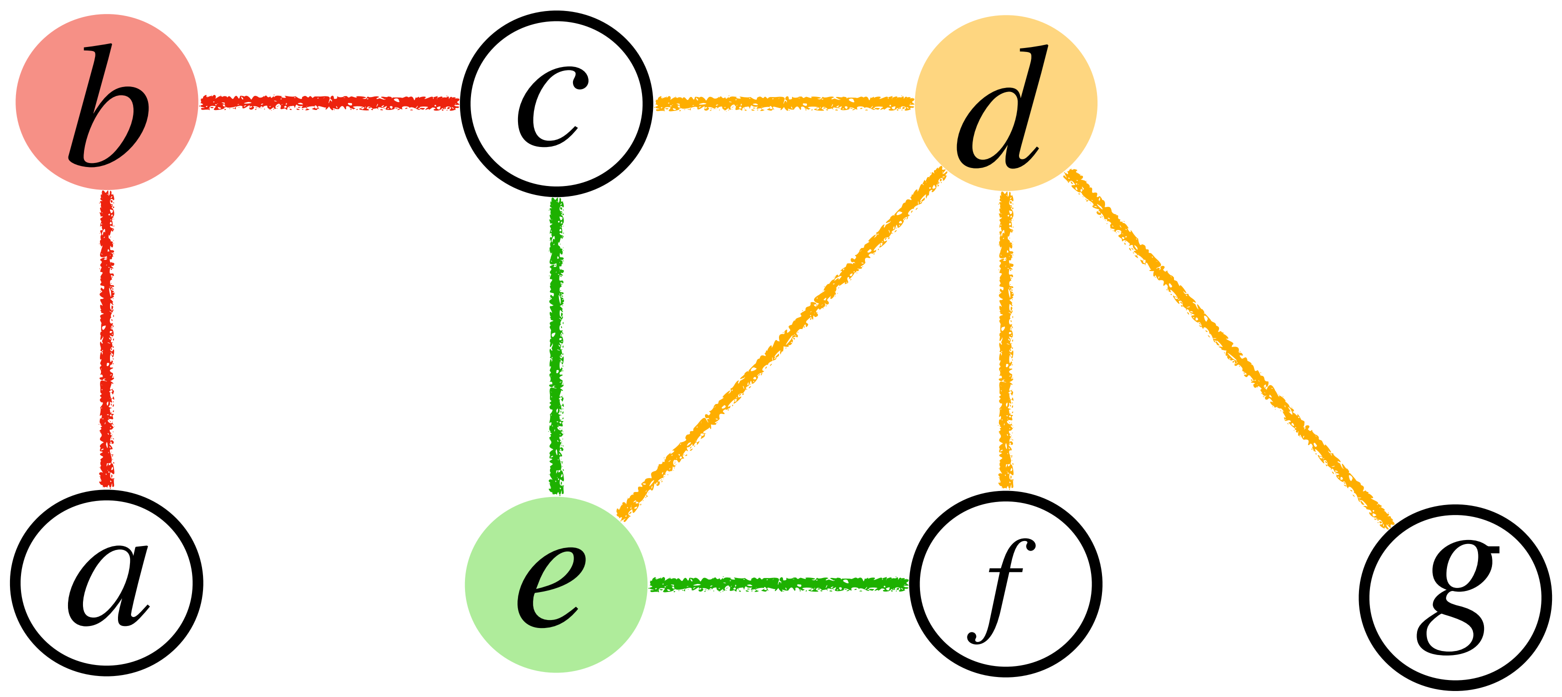
Example:



$$S = \{b, c, e, f, d, g\}$$

$$C = |S| = 6$$

$$C^* = 3$$



$$\frac{C}{C^*} = 2$$

Algorithm:

1. $S = \emptyset$ — S is the vertex cover.
2. $E' = E$ — E' are the uncovered edges.
3. while $E' \neq \emptyset$
4. let (u, v) be any edge in E'
5. $S \leftarrow S \cup \{u, v\}$
6. Remove all the edges that have either u or v as endpoints from E'
7. Return S

Quiz questions:

1. What is the main idea of the above approximation algorithm for the "Vertex Cover Problem"?

2. Can you think of an instance for which the above algorithm outputs an optimal solution, and an instance for which it outputs a non-optimal solution?

Roadmap of this lecture:

1. Define "Approximation Algorithm".

2. Understand approximation algorithms by solving the "Vertex Cover Problem".

    2.1 An approximation algorithm for "Vertex Cover Problem".

    2.2 Analyze the approximation ratio of the algorithm.

3. Understand approximation algorithms by solving the "Traveling Salesman Problem (TSP)".

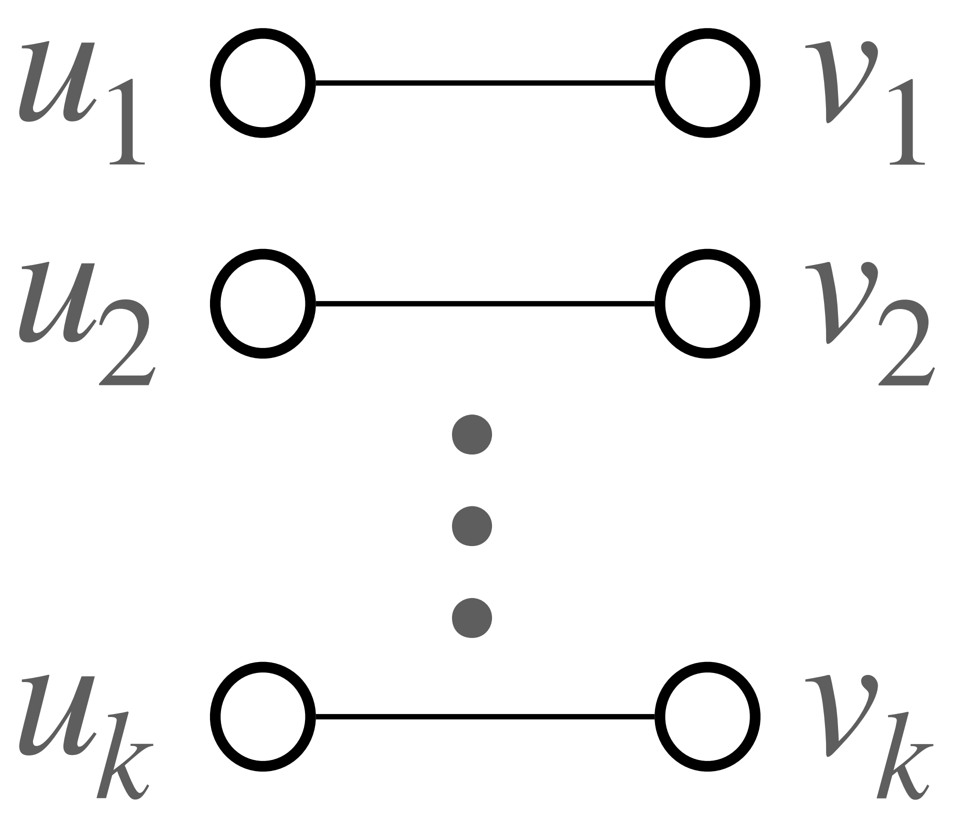
    3.1 An approximation algorithm for TSP with the triangle inequality.

    3.2 Analyze the approximation ratio of the algorithm.

Theorem: The algorithm for the Vertex Cover Problem is a polynomial-time 2-approximation algorithm.

Theorem: The algorithm for the Vertex Cover Problem is a polynomial-time 2-approximation algorithm.

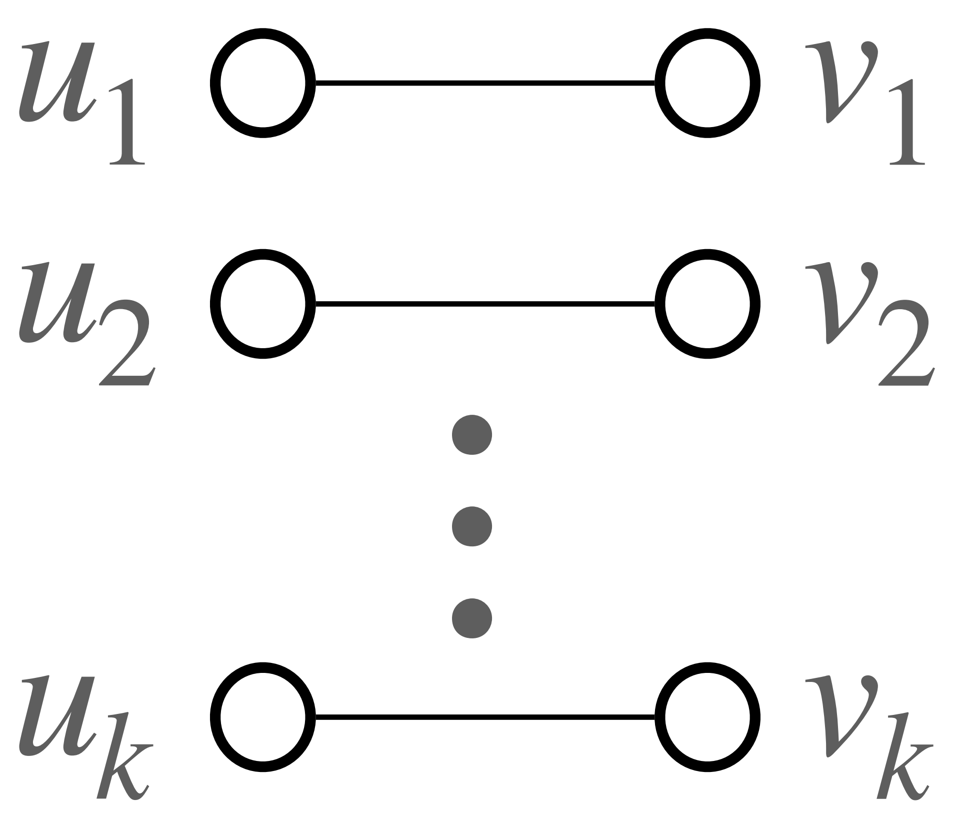
Proof: Let



be the k edges chosen by the algorithm.

Theorem: The algorithm for the Vertex Cover Problem is a polynomial-time 2-approximation algorithm.

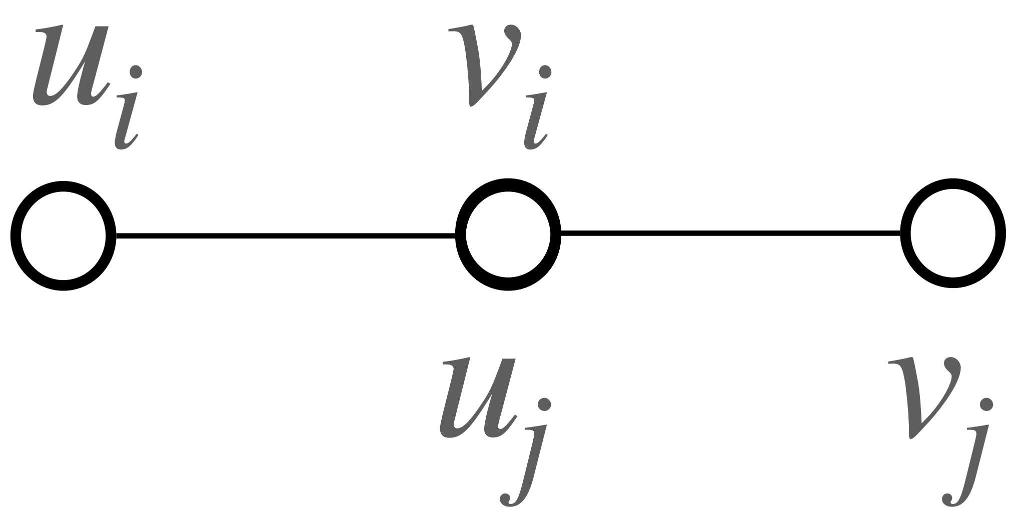
Proof: Let



be the k edges chosen by the algorithm.

No two edges above share any common node.

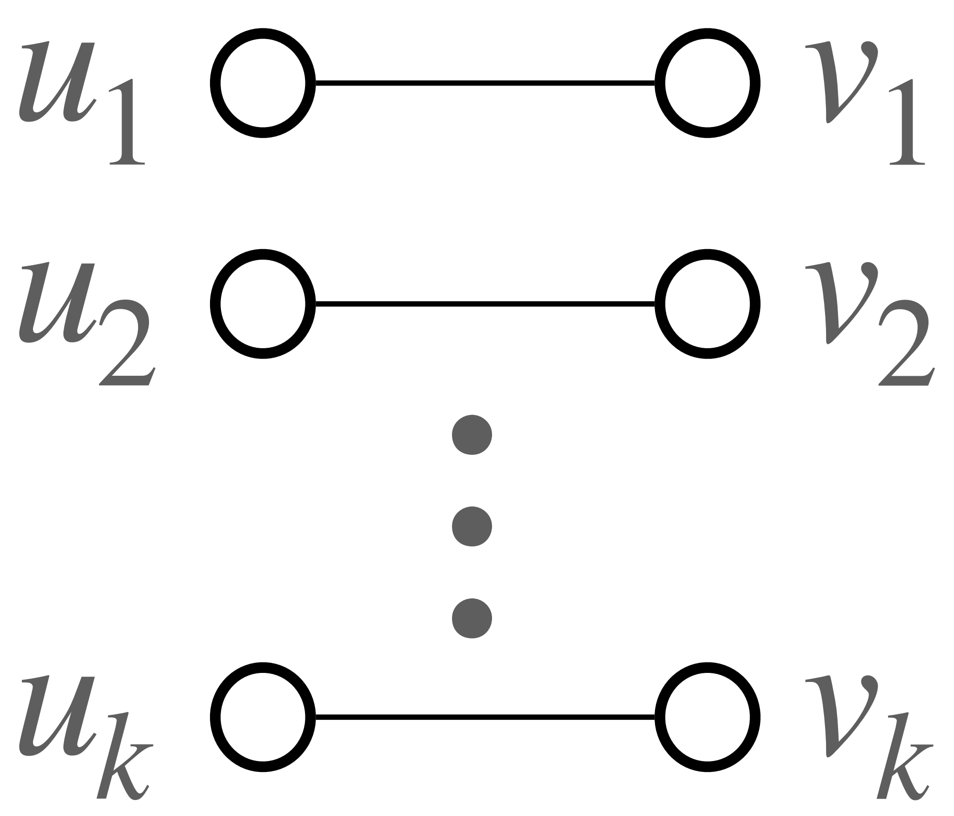
Impossible:



Why ?

Theorem: The algorithm for the Vertex Cover Problem is a polynomial-time 2-approximation algorithm.

Proof: Let



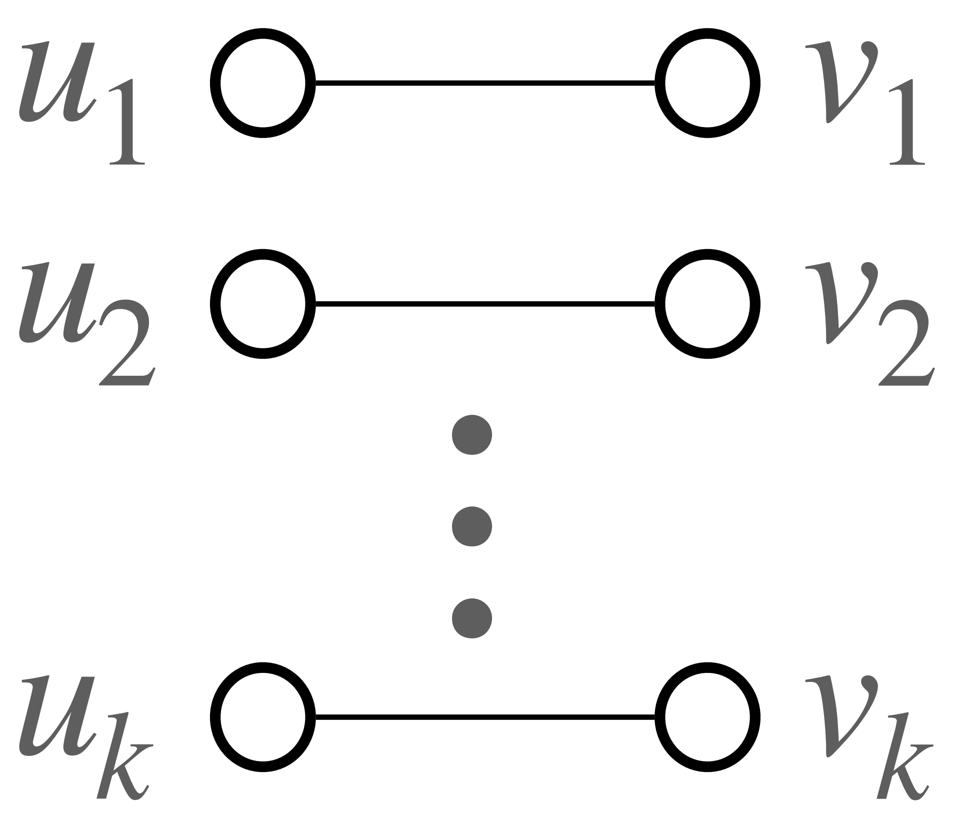
be the k edges chosen by the algorithm.

No two edges above share any common node.

For each of those k disjoint edges, say edge (u_i, v_i), every vertex cover needs to choose either u_i or v_i .

Theorem: The algorithm for the Vertex Cover Problem is a polynomial-time 2-approximation algorithm.

Proof: Let



be the k edges chosen by the algorithm.

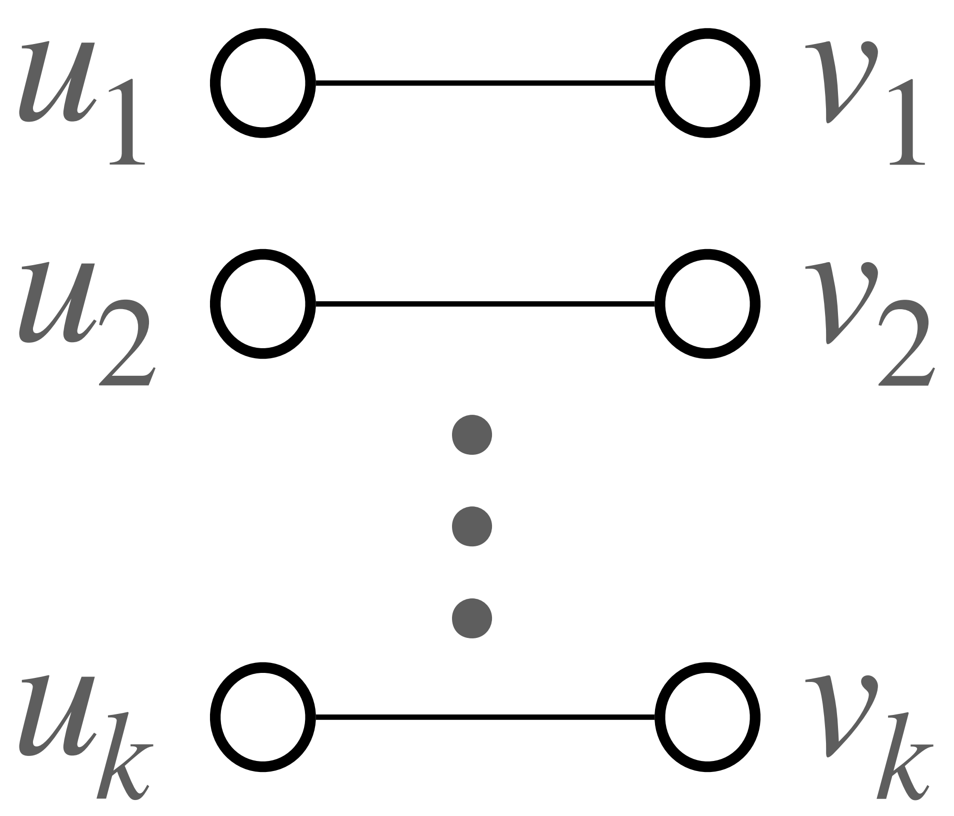
No two edges above share any common node.

For each of those k disjoint edges, say edge (u_i, v_i), every vertex cover needs to choose either u_i or v_i .

So $C^* \geq k$

Theorem: The algorithm for the Vertex Cover Problem is a polynomial-time 2-approximation algorithm.

Proof: Let



be the k edges chosen by the algorithm.

No two edges above share any common node.

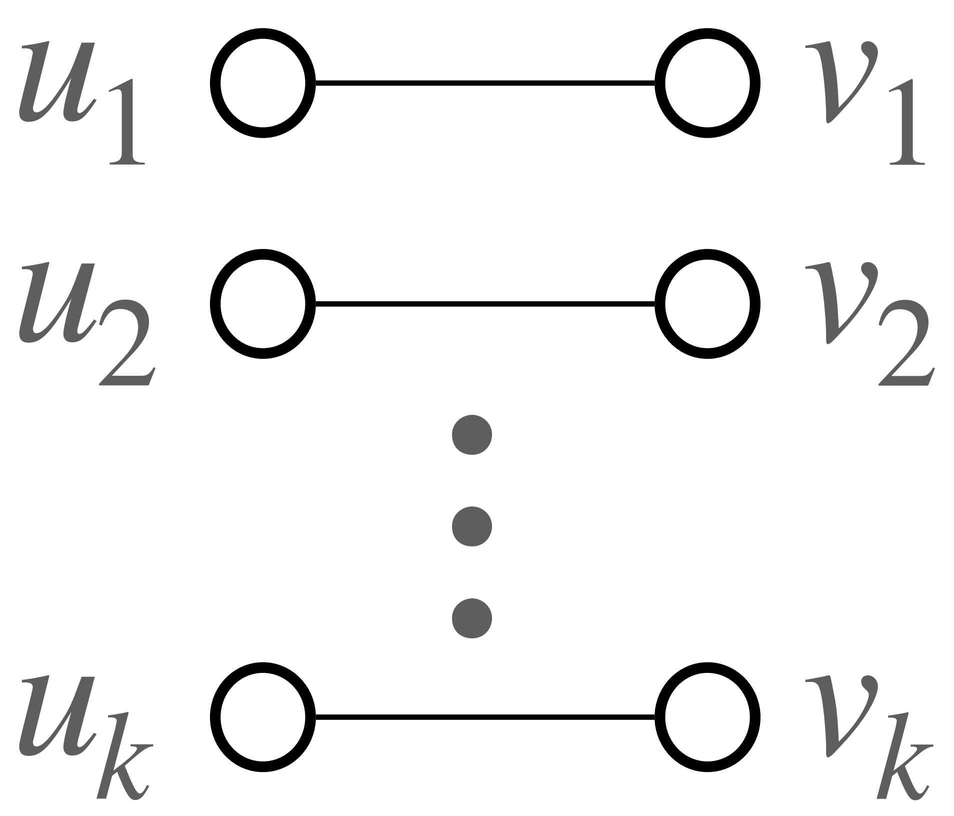
For each of those k disjoint edges, say edge (u_i, v_i), every vertex cover needs to choose either u_i or v_i .

$\boxed{\text{So } \quad C^* \geq k}$

As $\quad C = 2k$

Theorem: The algorithm for the Vertex Cover Problem
is a polynomial-time
2-approximation algorithm.

Proof: Let



be the

k edges chosen by the algorithm.

No two edges above share any common node.

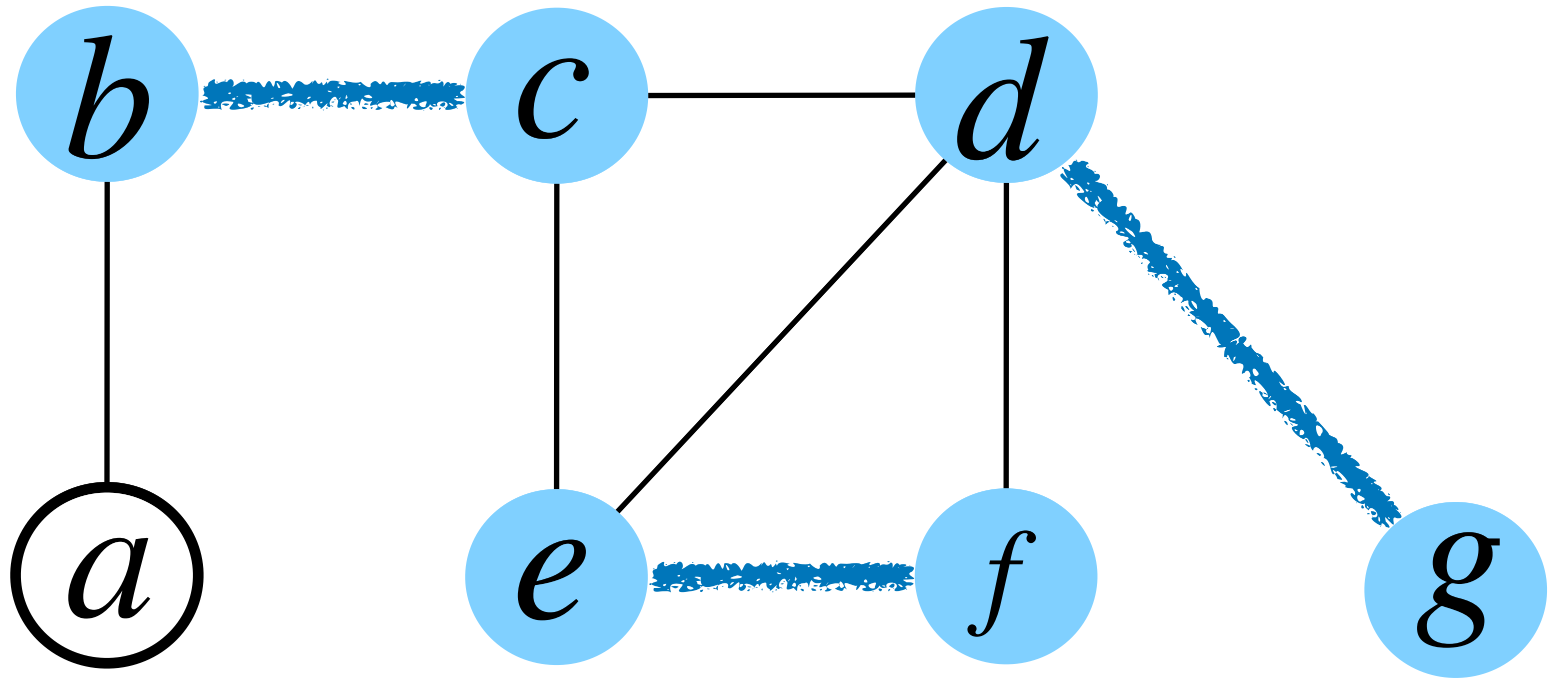
For each of those k disjoint edges,
say edge (u_i, v_i),
every vertex cover needs to choose either u_i or v_i .

$$\boxed{\text{So} \quad C^* \geq k}$$

As $C = 2k$

$$\boxed{\frac{C}{C^*} \leq \frac{2k}{k} = 2}$$

Let's take a look at our earlier example:

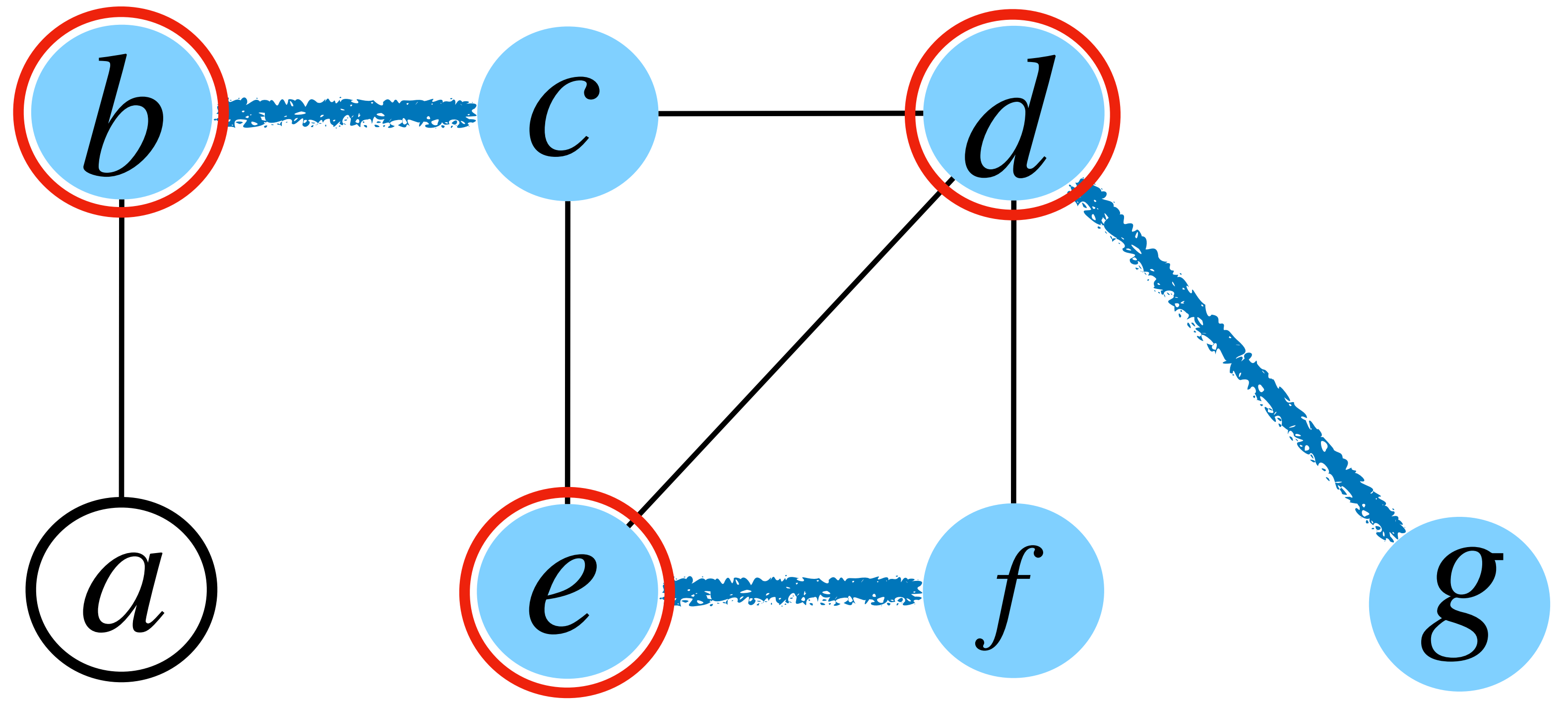


3 disjoint chosen edges

$$S = \{b, c, e, f, d, g\}$$

$$C = |S| = 6$$

Let's take a look at our earlier example:



$$S = \{b, c, e, f, d, g\}$$

$$C = |S| = 6$$

3 disjoint chosen edges

Every vertex cover will contain at least half of those endpoints.

So the approximation ratio is 2.

How to prove $\frac{C}{C^*} \leq \rho$ without knowing C^* ?

For minimization problem, we just need a lower bound of C^*.

Remember our earlier proof:

$\boxed{\text{So } C^* \geq k}$ Lower bound

As $C = 2k$

$$\boxed{\frac{C}{C^*} \leq \frac{2k}{k} = 2}$$

How to prove $\frac{C}{C^*} \leq \rho$ without knowing C^* ?

maximization **upper bound**

For minimization problem, we just need a lower bound of C^* .

Remember our earlier proof:

$\boxed{\text{So } C^* \geq k}$ Lower bound

As $C = 2k$

$$\boxed{\frac{C}{C^*} \leq \frac{2k}{k} = 2}$$

$$\boxed{\frac{C^*}{C} \leq \frac{\text{upper bound of } C^*}{C}}$$

Quiz questions:

1. For the above approximation algorithm for the "Vertex Cover Problem", how did we find the approximation ratio without knowing the optimal cost?

2. The "Vertex Cover Problem" is a minimization problem. For a maximization problem, how can we find the approximation ratio without knowing the optimal cost?

Roadmap of this lecture:

1. Define "Approximation Algorithm".

2. Understand approximation algorithms by solving the "Vertex Cover Problem".

    2.1 An approximation algorithm for "Vertex Cover Problem".

    2.2 Analyze the approximation ratio of the algorithm.

3. Understand approximation algorithms by solving the "Traveling Salesman Problem (TSP)".

    3.1 An approximation algorithm for TSP with the triangle inequality.

    3.2 Analyze the approximation ratio of the algorithm.

Traveling Salesman Problem (TSP)

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.

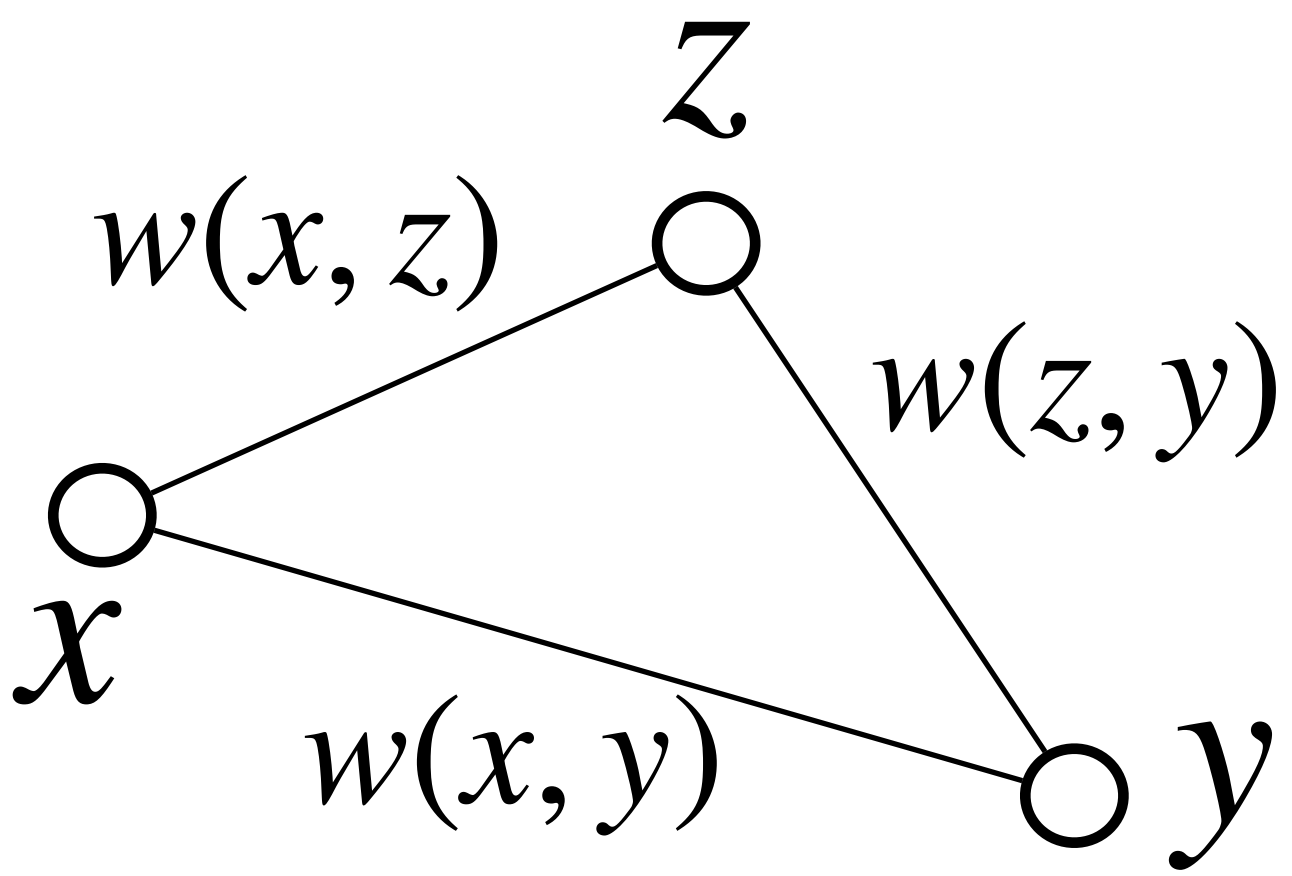
Output: A Hamiltonian cycle of minimum weight.

Traveling Salesman Problem (TSP)

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.

Output: A Hamiltonian cycle of minimum weight.

Triangle Inequality:



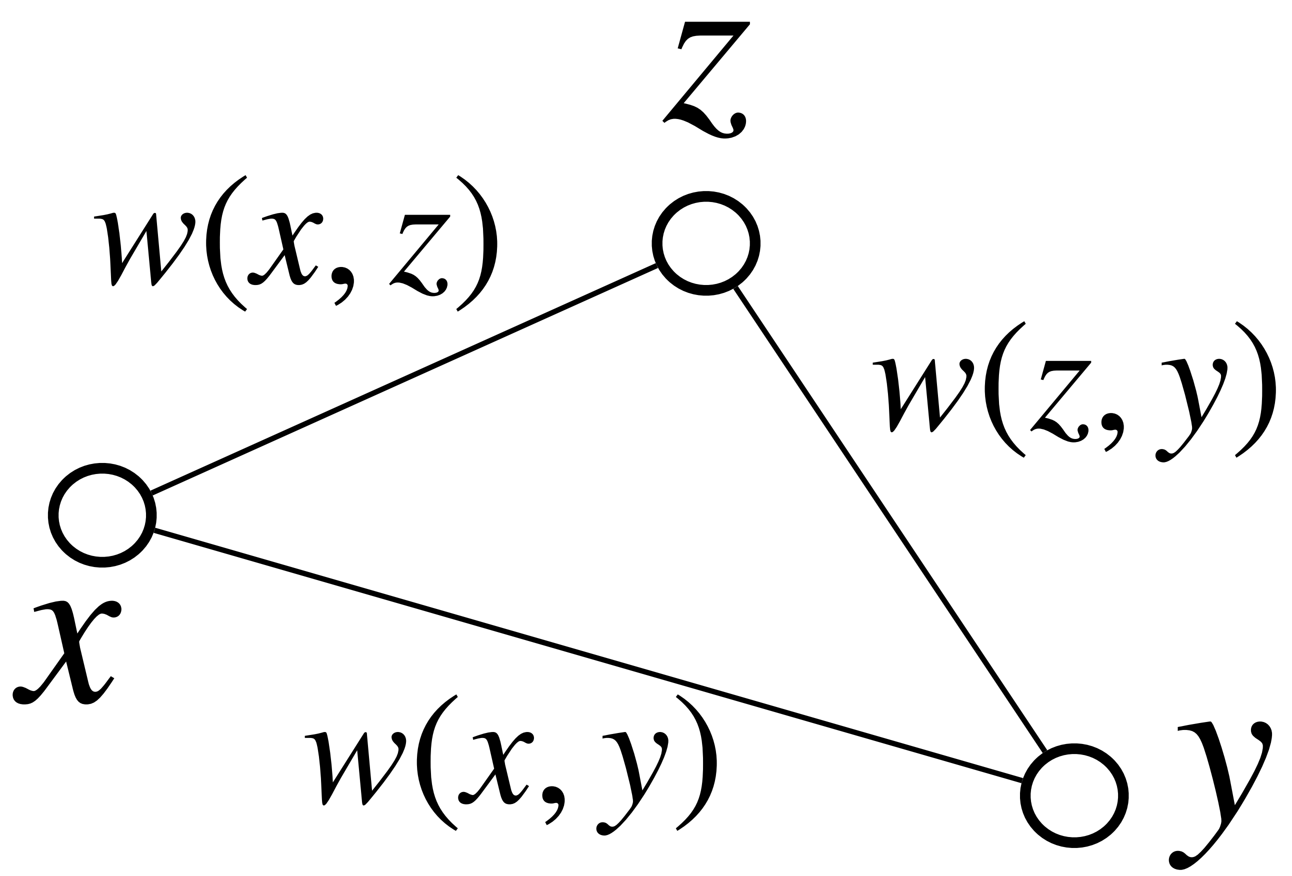
$$w(x, y) \leq w(x, z) + w(z, y)$$

Traveling Salesman Problem (TSP) with the Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$. The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Triangle Inequality:



$$w(x, y) \leq w(x, z) + w(z, y)$$

TSP with Triangle Inequality

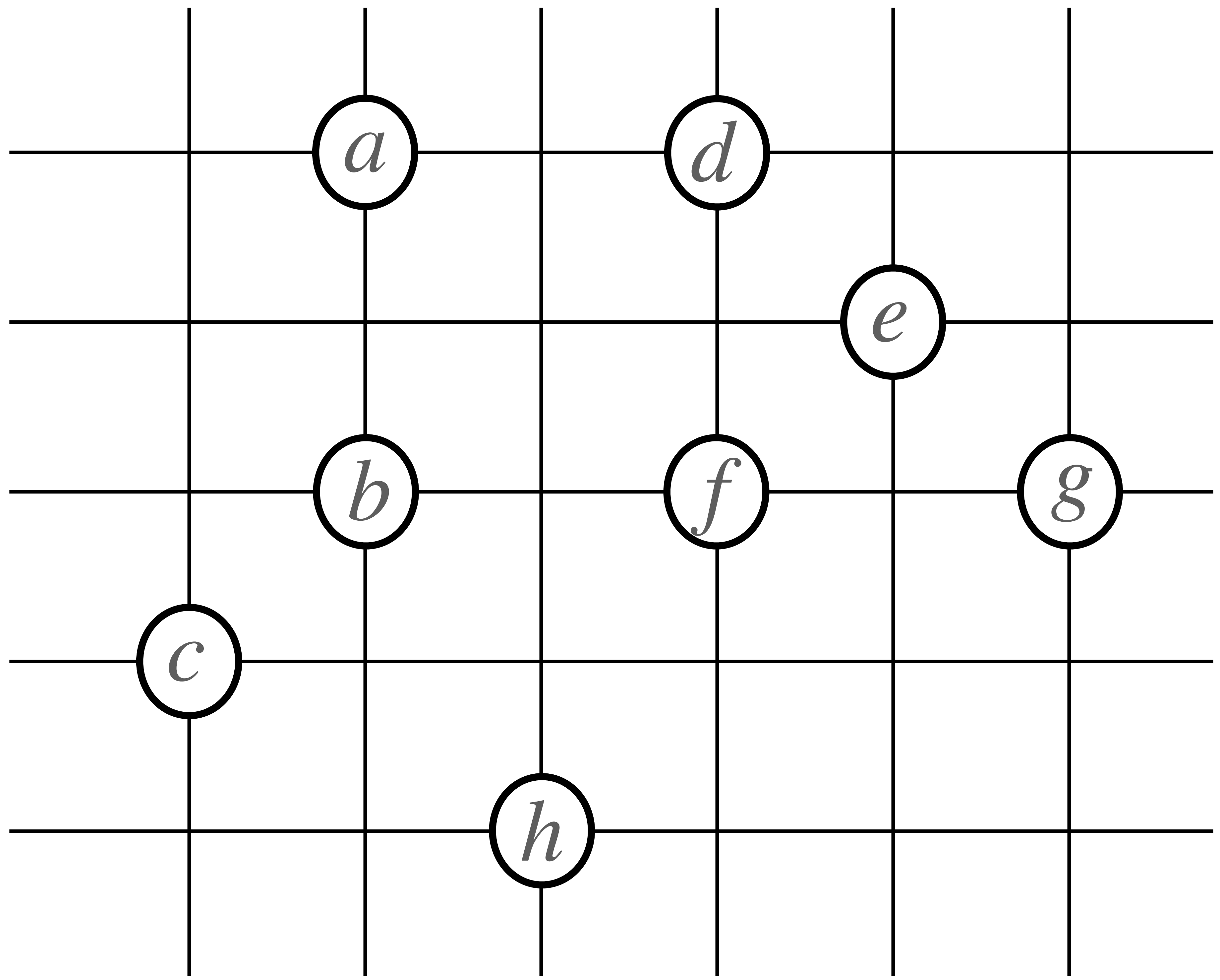
Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$. The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

Edge weight between every two nodes: Their Euclidean distance.



TSP with Triangle Inequality

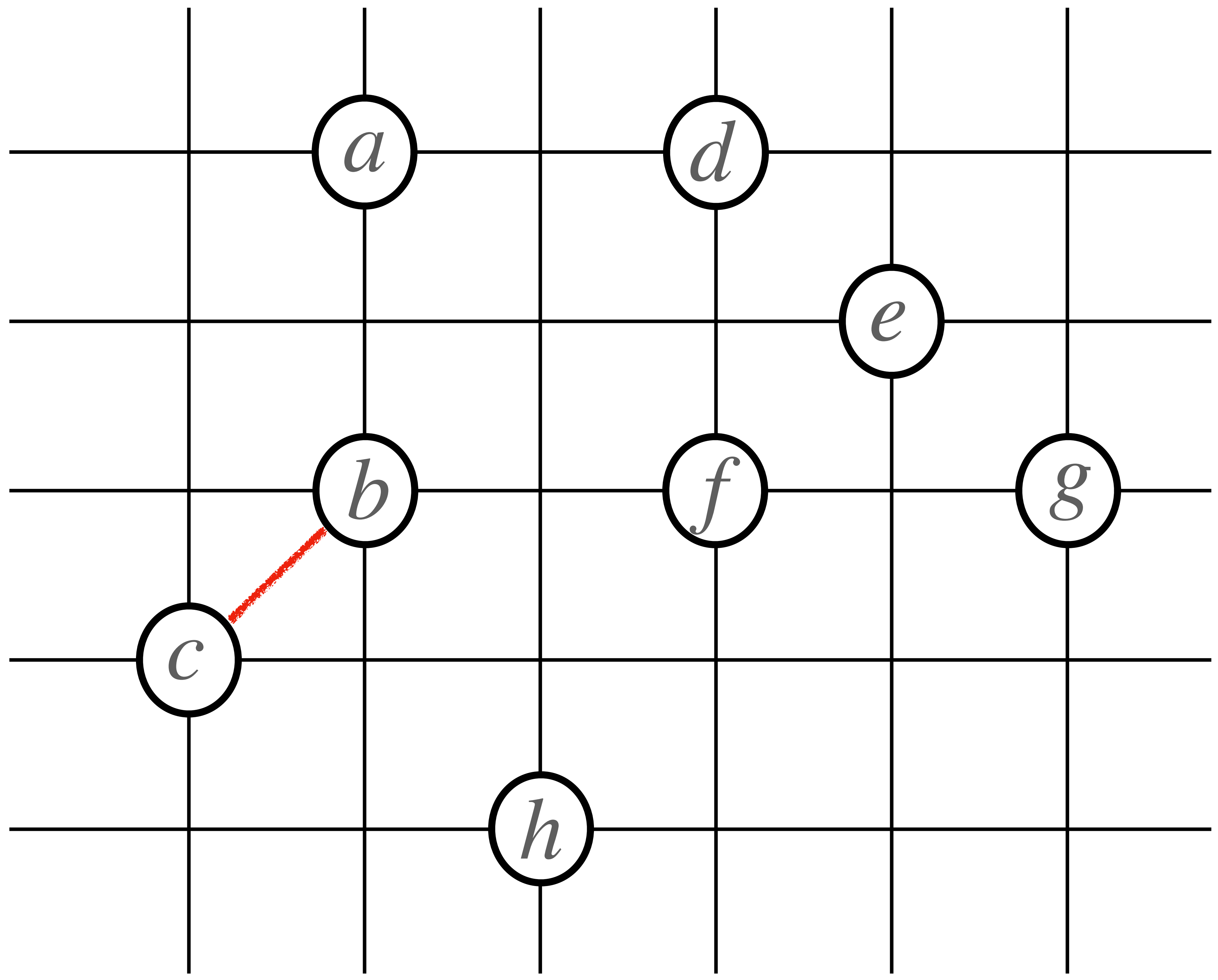
Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.
The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

Edge weight between every two nodes: Their Euclidean distance.

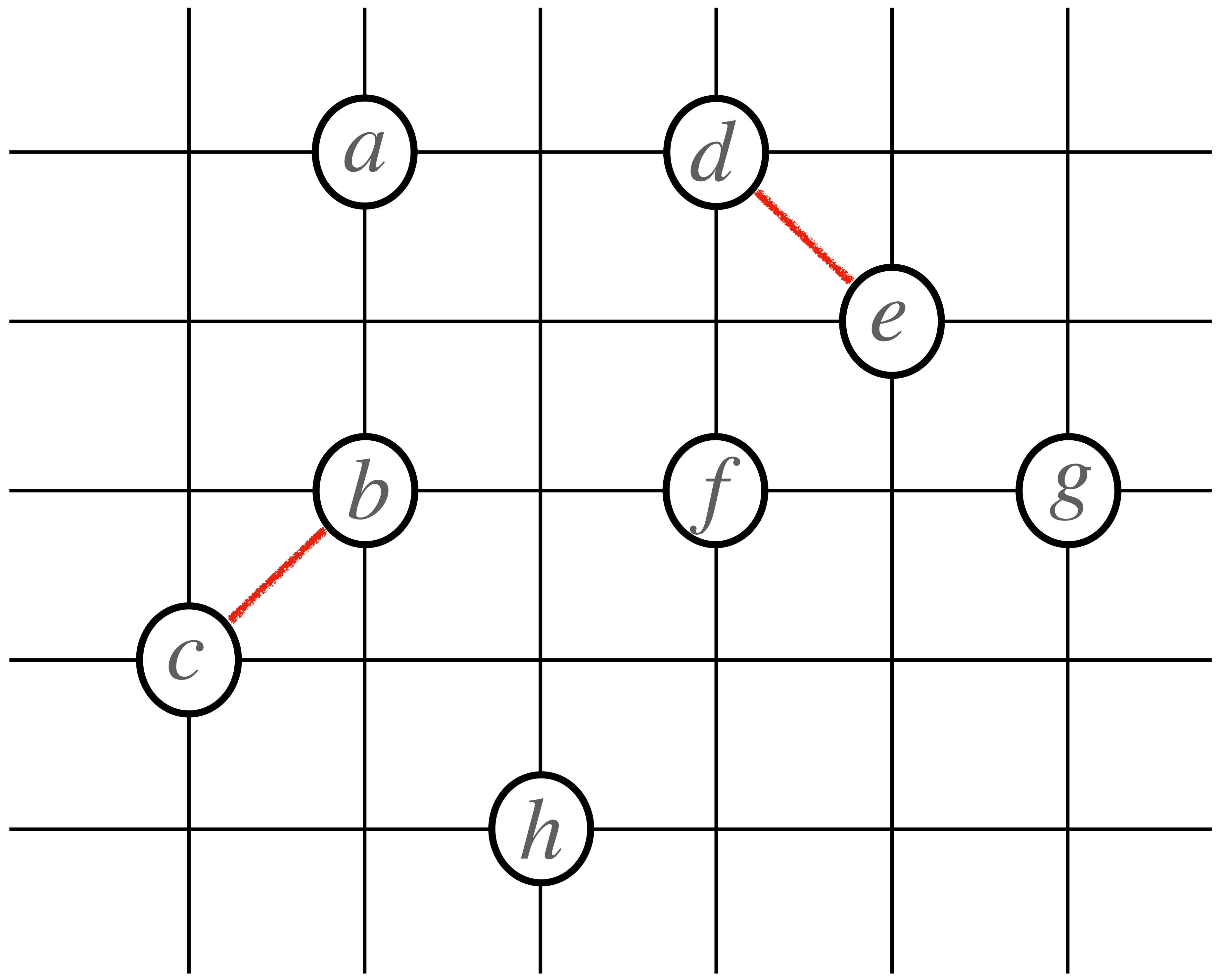


TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u,v) \in E$ has a non-negative integer weight $w(u,v)$.
The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Edge weight between every two nodes: Their Euclidean distance.



Idea of Algorithm:

1. Build an MST of G.

TSP with Triangle Inequality

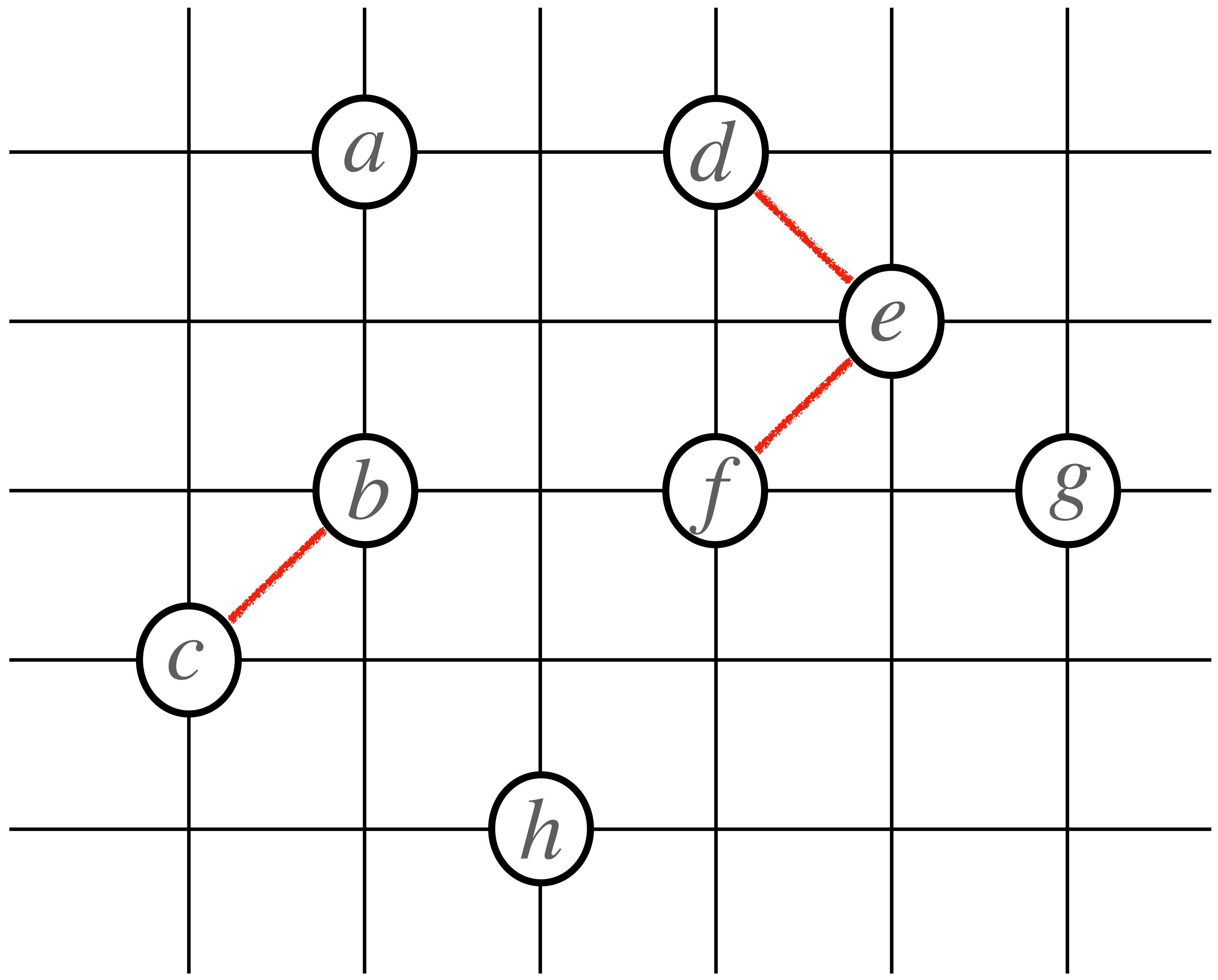
Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.
The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

Edge weight between every two nodes: Their Euclidean distance.



TSP with Triangle Inequality

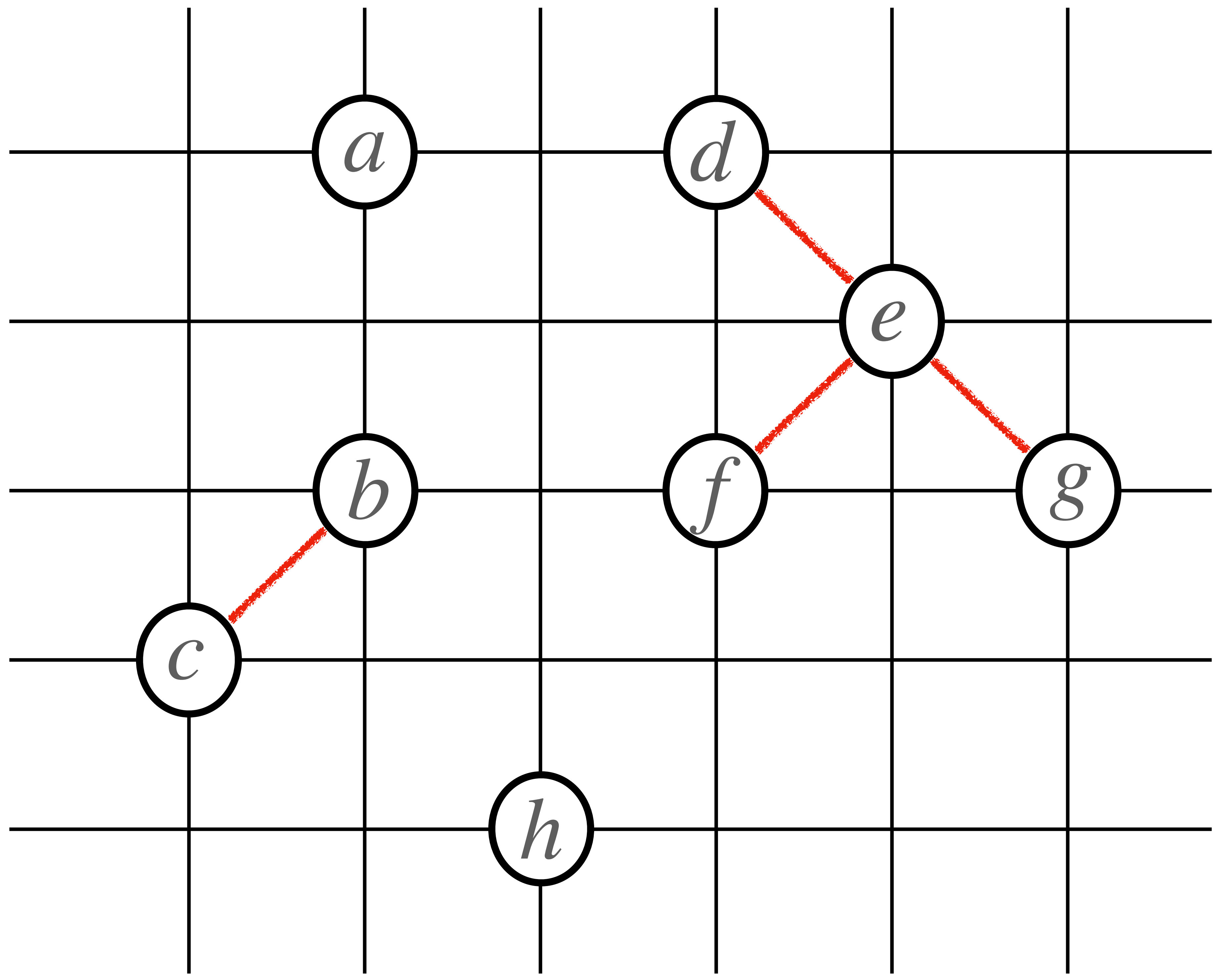
Input: An undirected complete graph G=(V,E), where every edge $(u,v) \in E$ has a non-negative integer weight $w(u,v)$.
The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Edge weight between every two nodes:
Their Euclidean distance.

Idea of Algorithm:

1. Build an MST of G.



TSP with Triangle Inequality

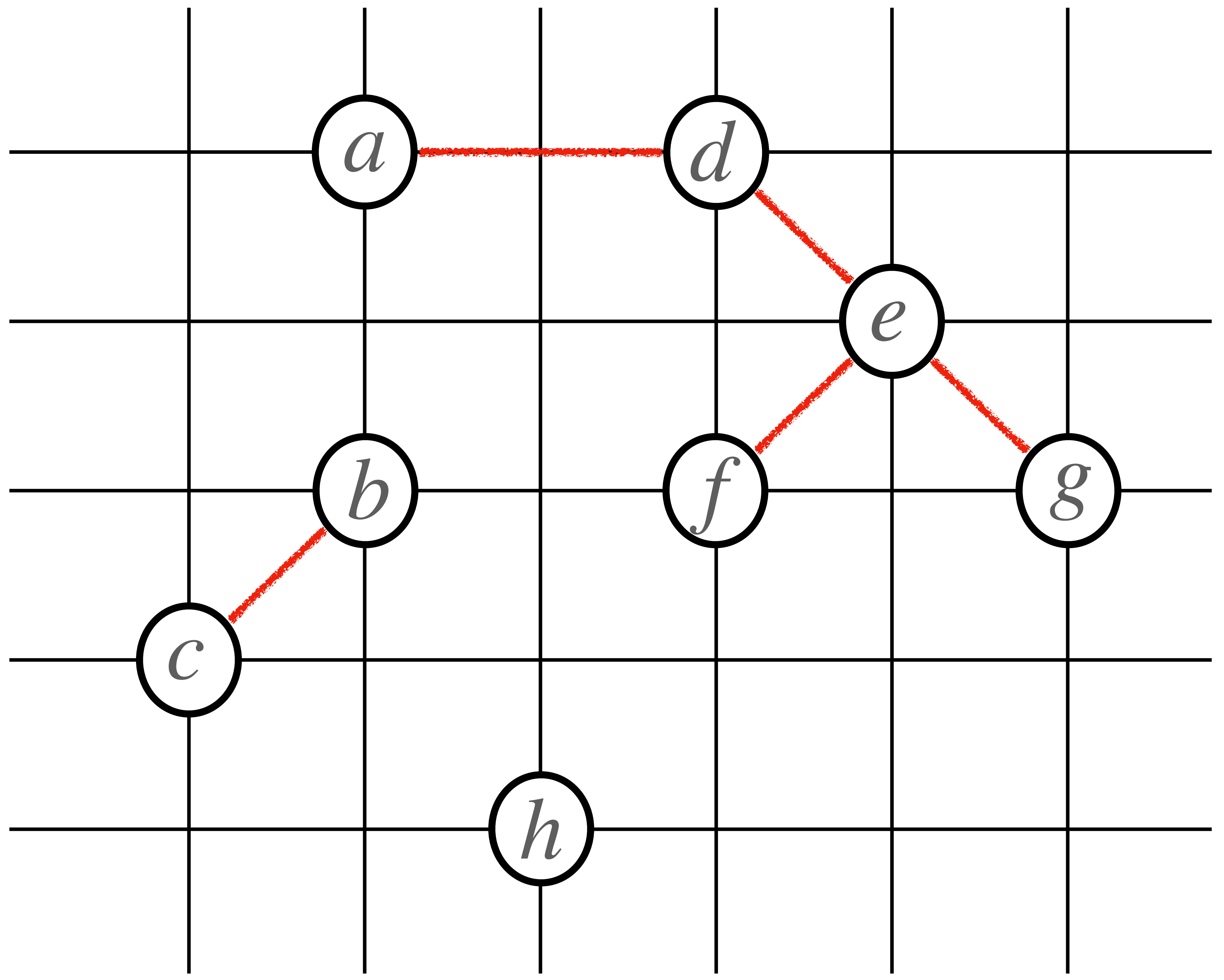
Input: An undirected complete graph G=(V,E), where every edge $(u,v) \in E$ has a non-negative integer weight $w(u,v)$. The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

Edge weight between every two nodes:
Their Euclidean distance.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.

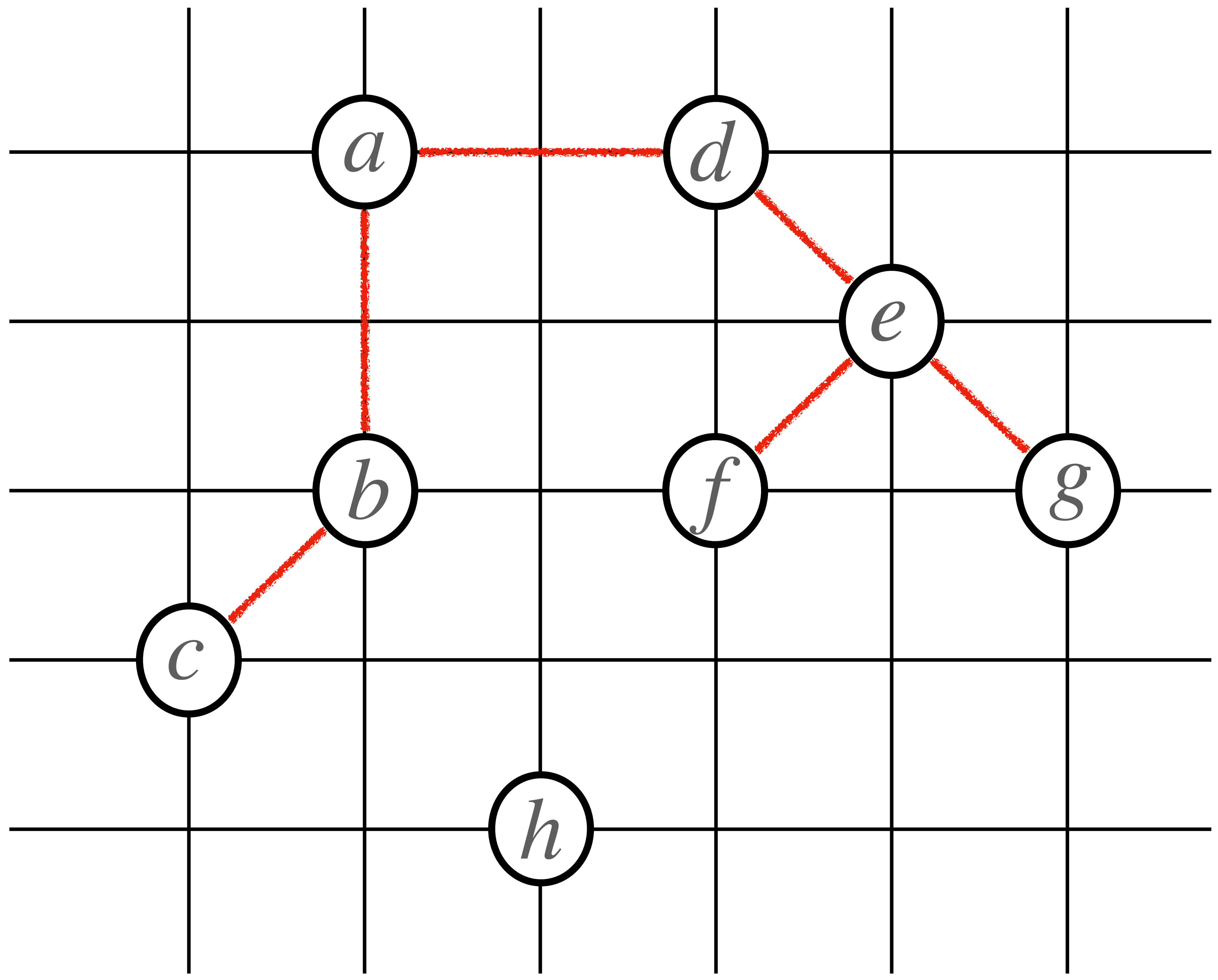
The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Edge weight between every two nodes: Their Euclidean distance.

Idea of Algorithm:

1. Build an MST of G.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.

The edge weights satisfy the triangle inequality.

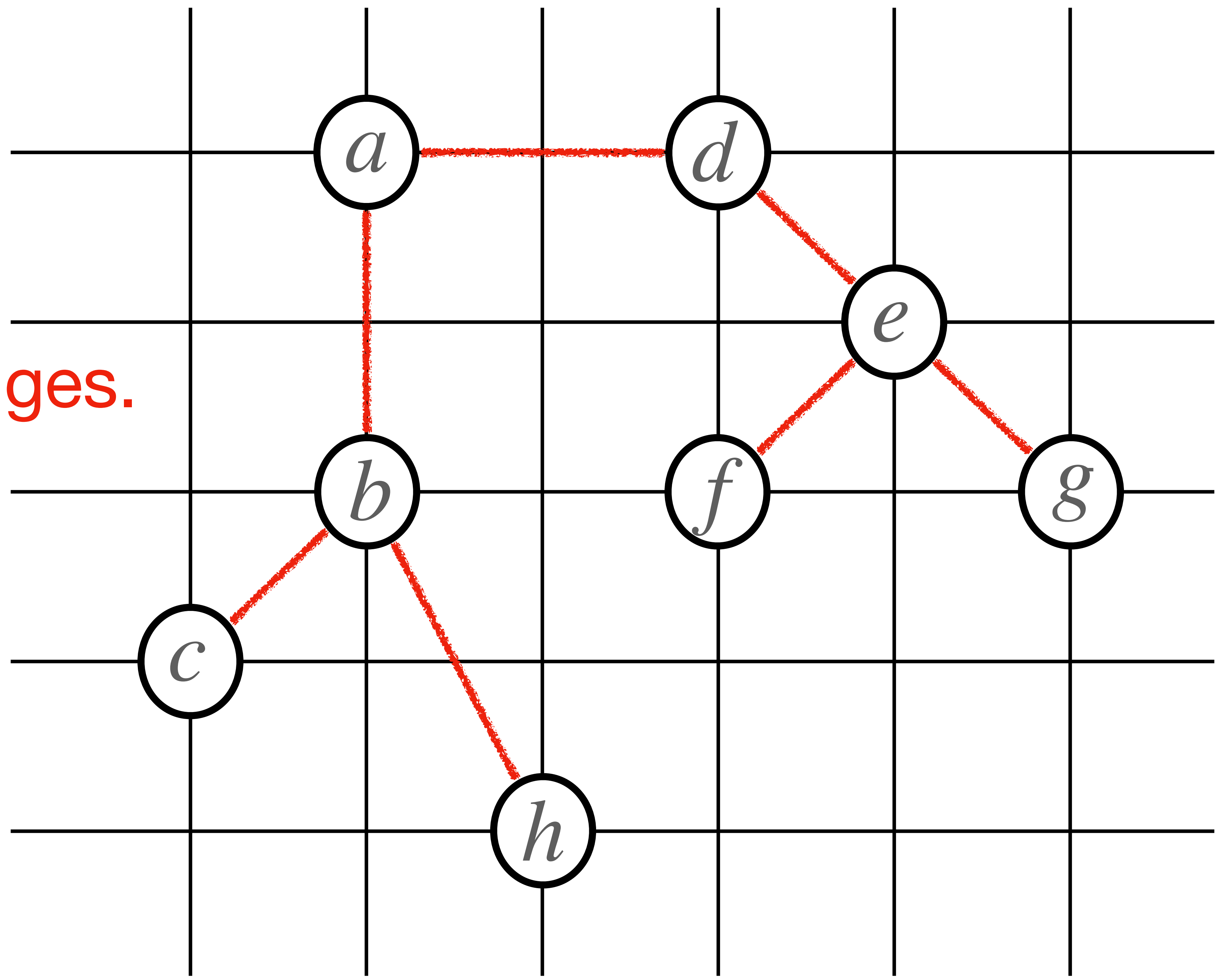
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

Edge weight between every two nodes: Their Euclidean distance.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.

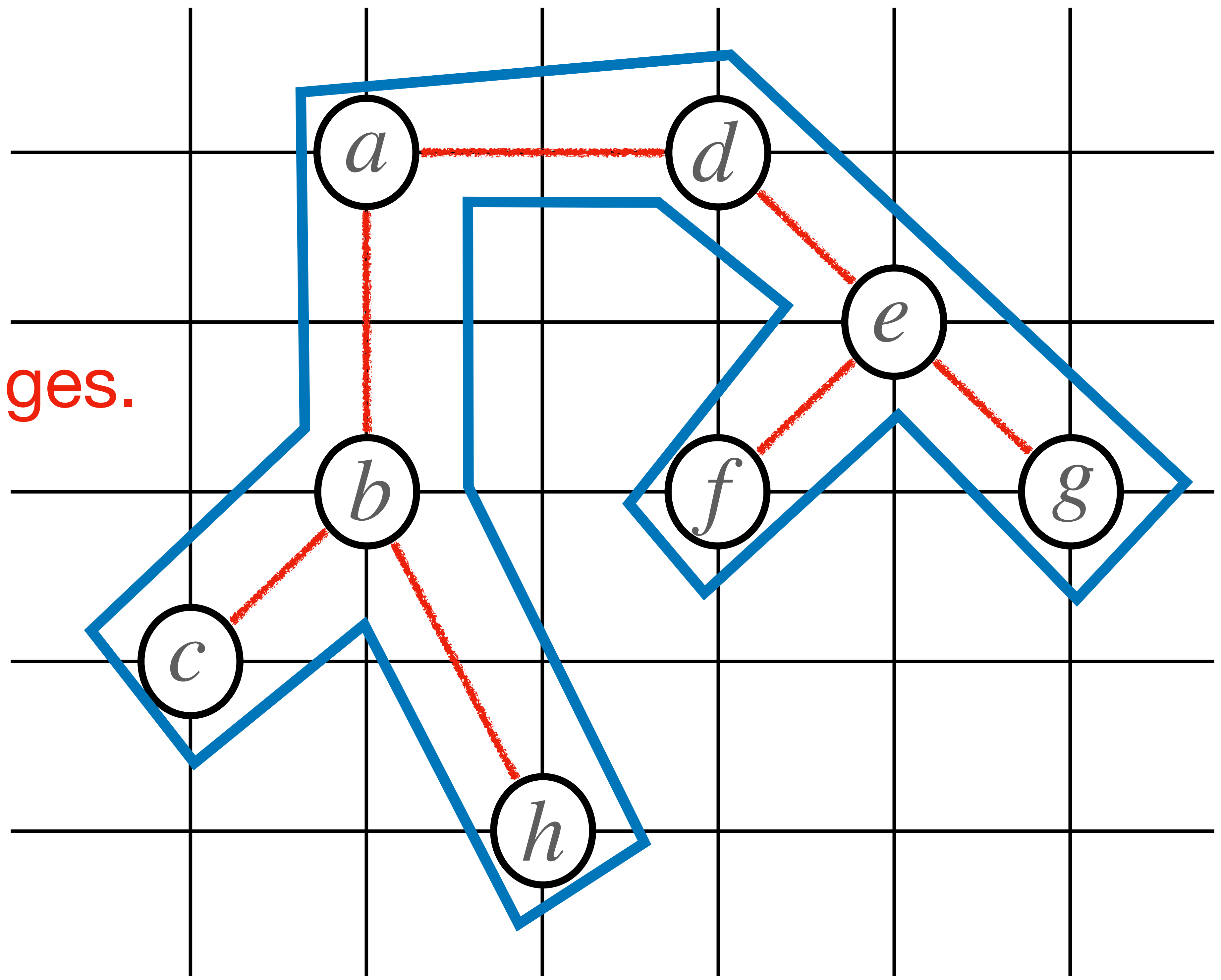
The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.

The edge weights satisfy the triangle inequality.

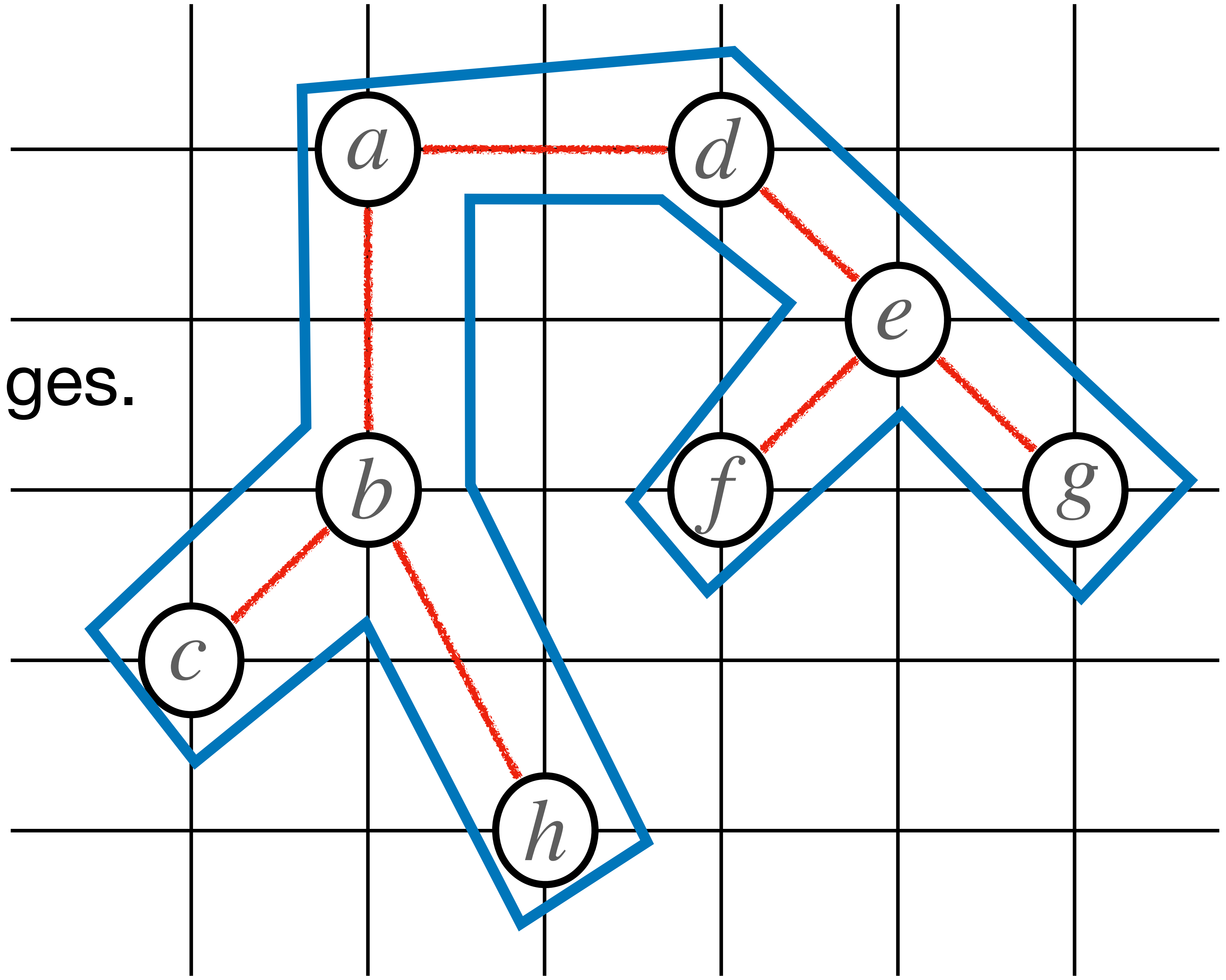
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

3. Get a Hamiltonian cycle by taking “short cuts”.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u,v) \in E$ has a non-negative integer weight $w(u,v)$.

The edge weights satisfy the triangle inequality.

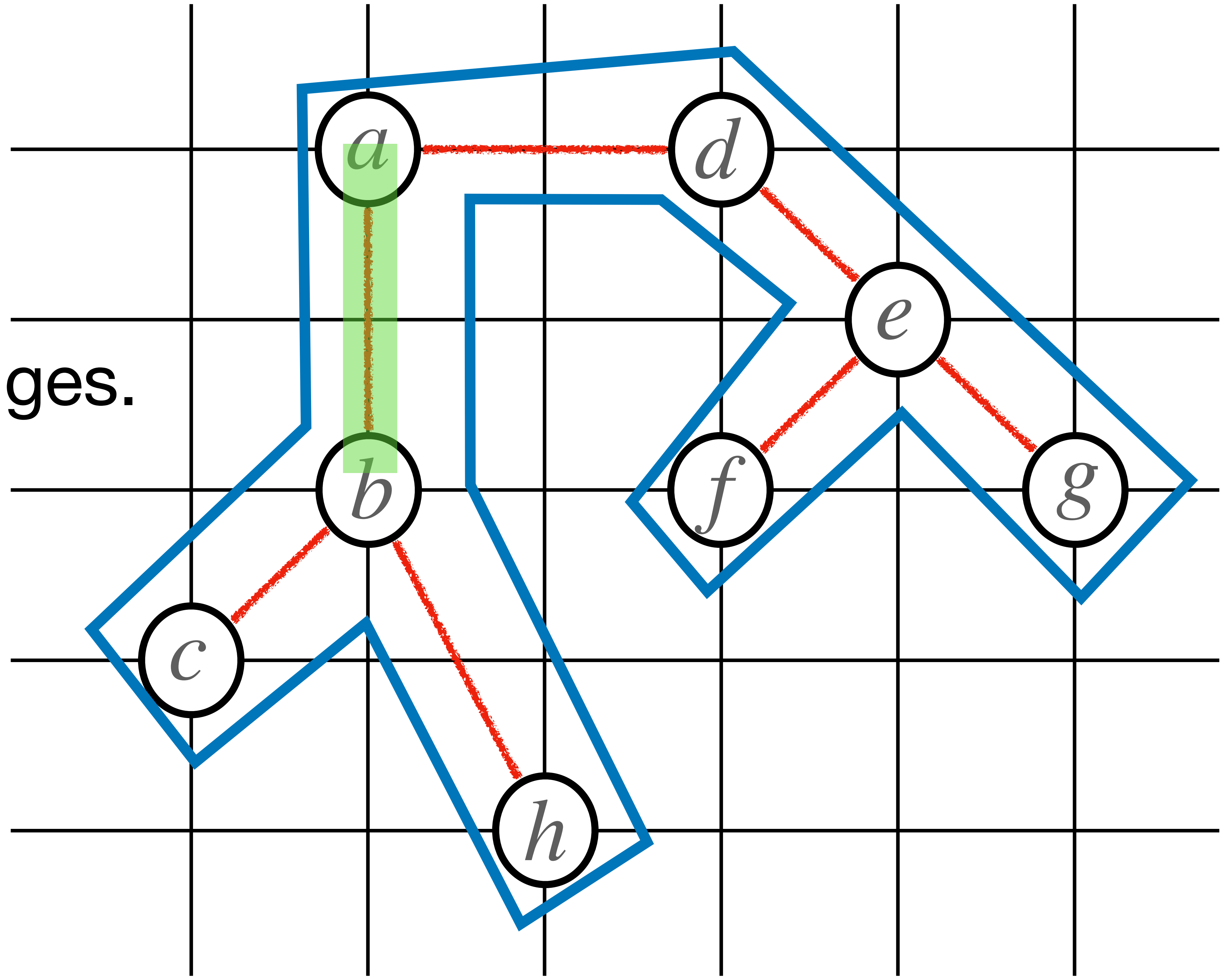
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

3. Get a Hamiltonian cycle by taking “short cuts”.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$. The edge weights satisfy the triangle inequality.

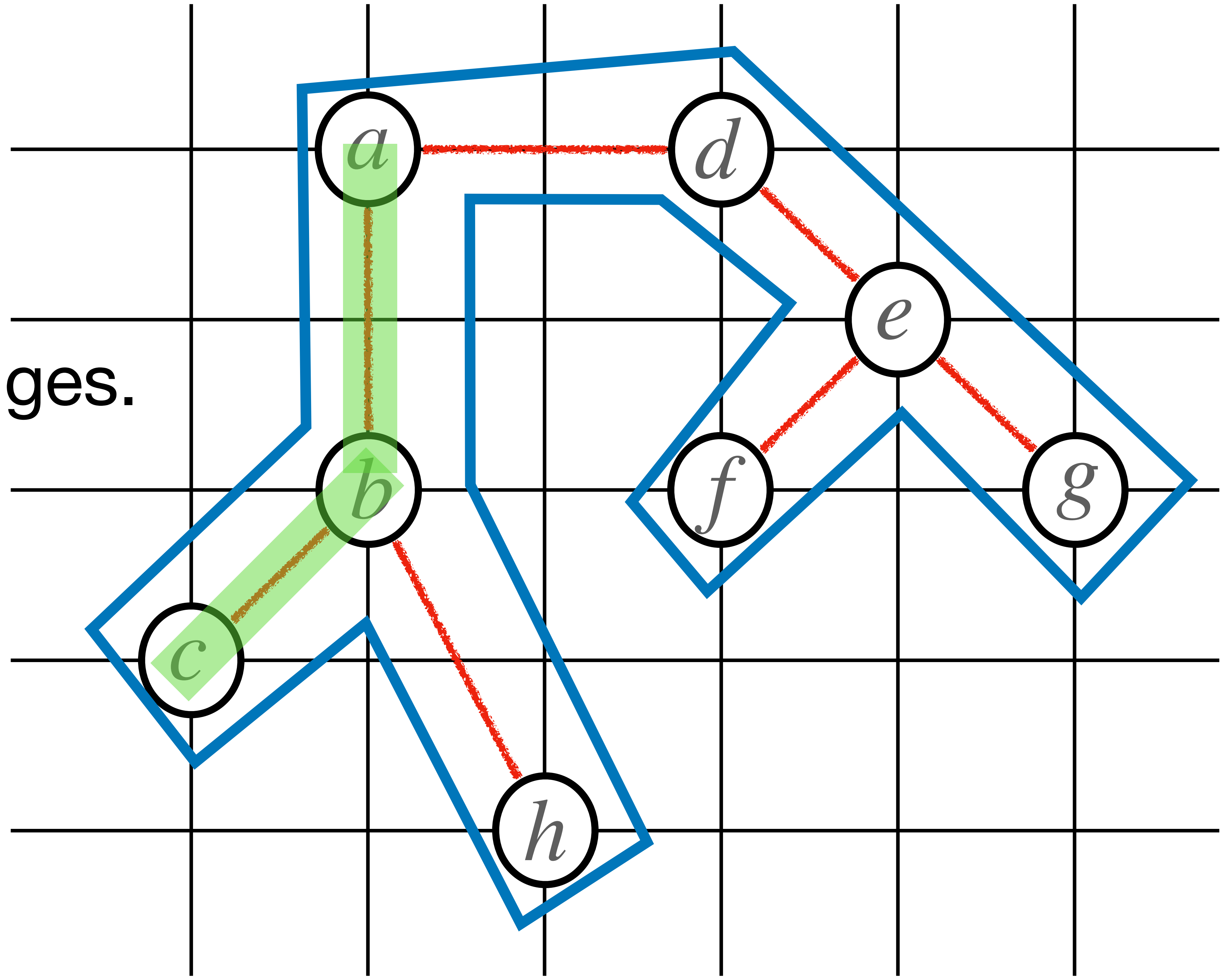
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

3. Get a Hamiltonian cycle by taking “short cuts”.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.
The edge weights satisfy the triangle inequality.

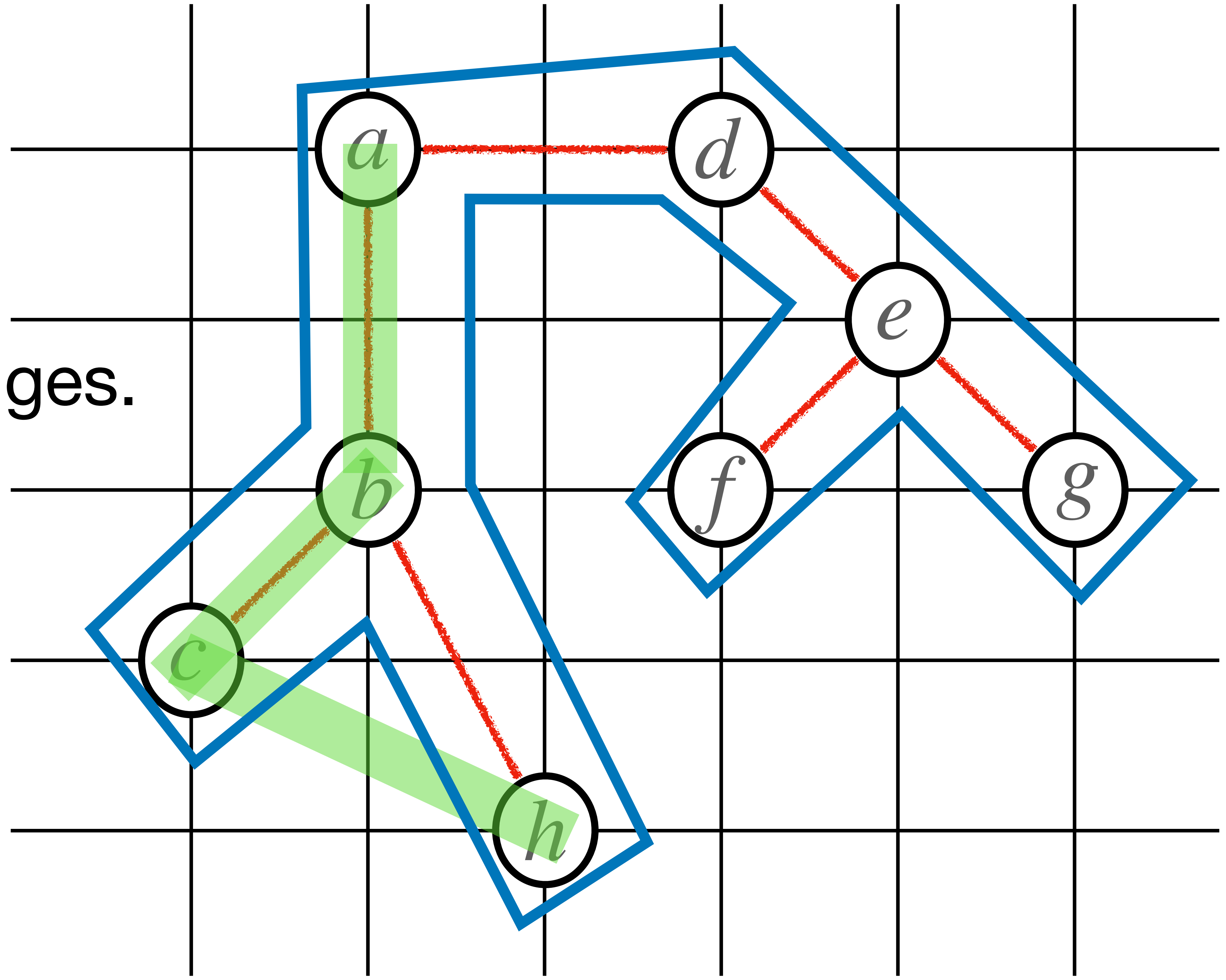
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

3. Get a Hamiltonian cycle by taking “short cuts”.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.
The edge weights satisfy the triangle inequality.

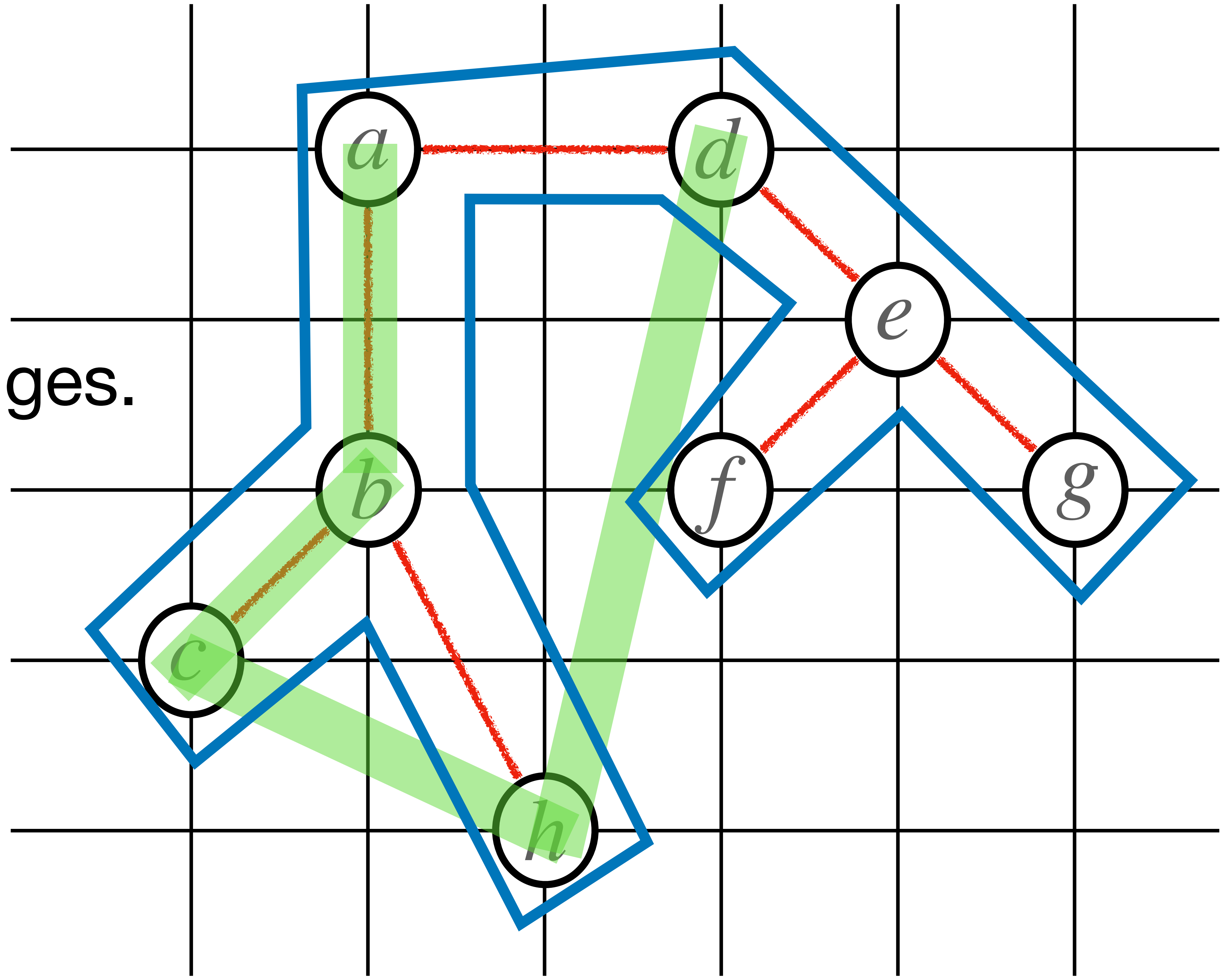
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

3. Get a Hamiltonian cycle by taking “short cuts”.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.
The edge weights satisfy the triangle inequality.

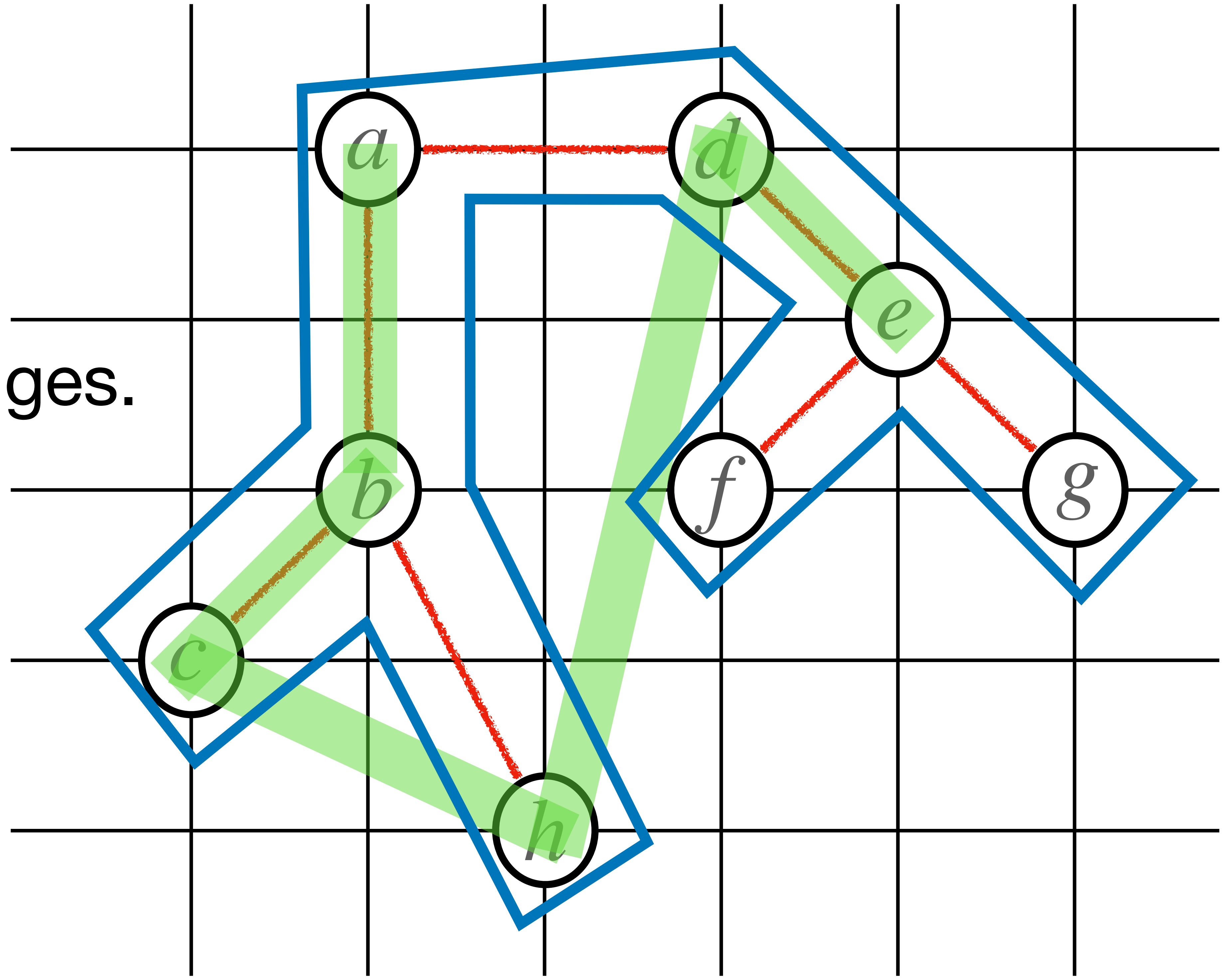
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

3. Get a Hamiltonian cycle by taking “short cuts”.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.
The edge weights satisfy the triangle inequality.

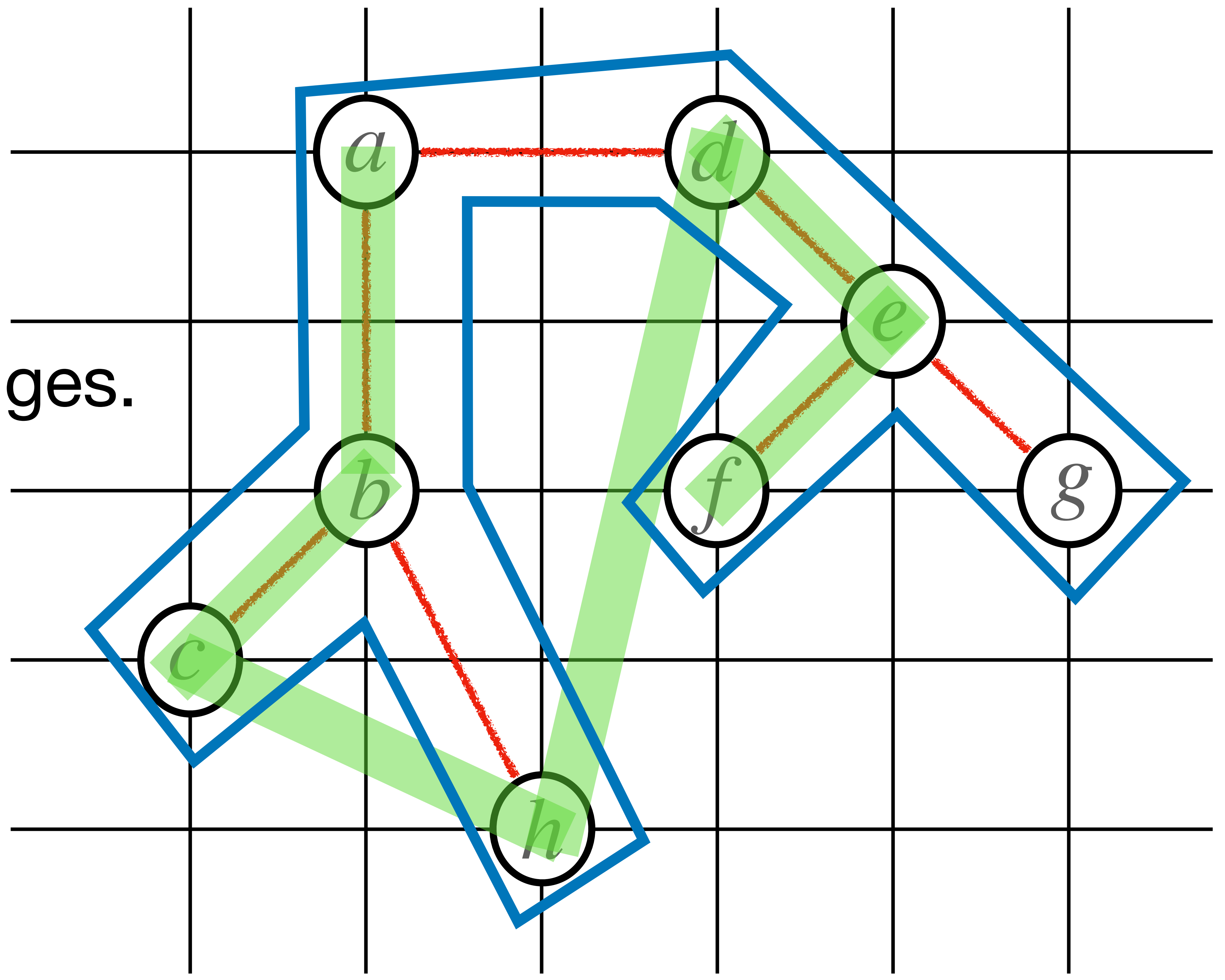
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

3. Get a Hamiltonian cycle by taking “short cuts”.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.
The edge weights satisfy the triangle inequality.

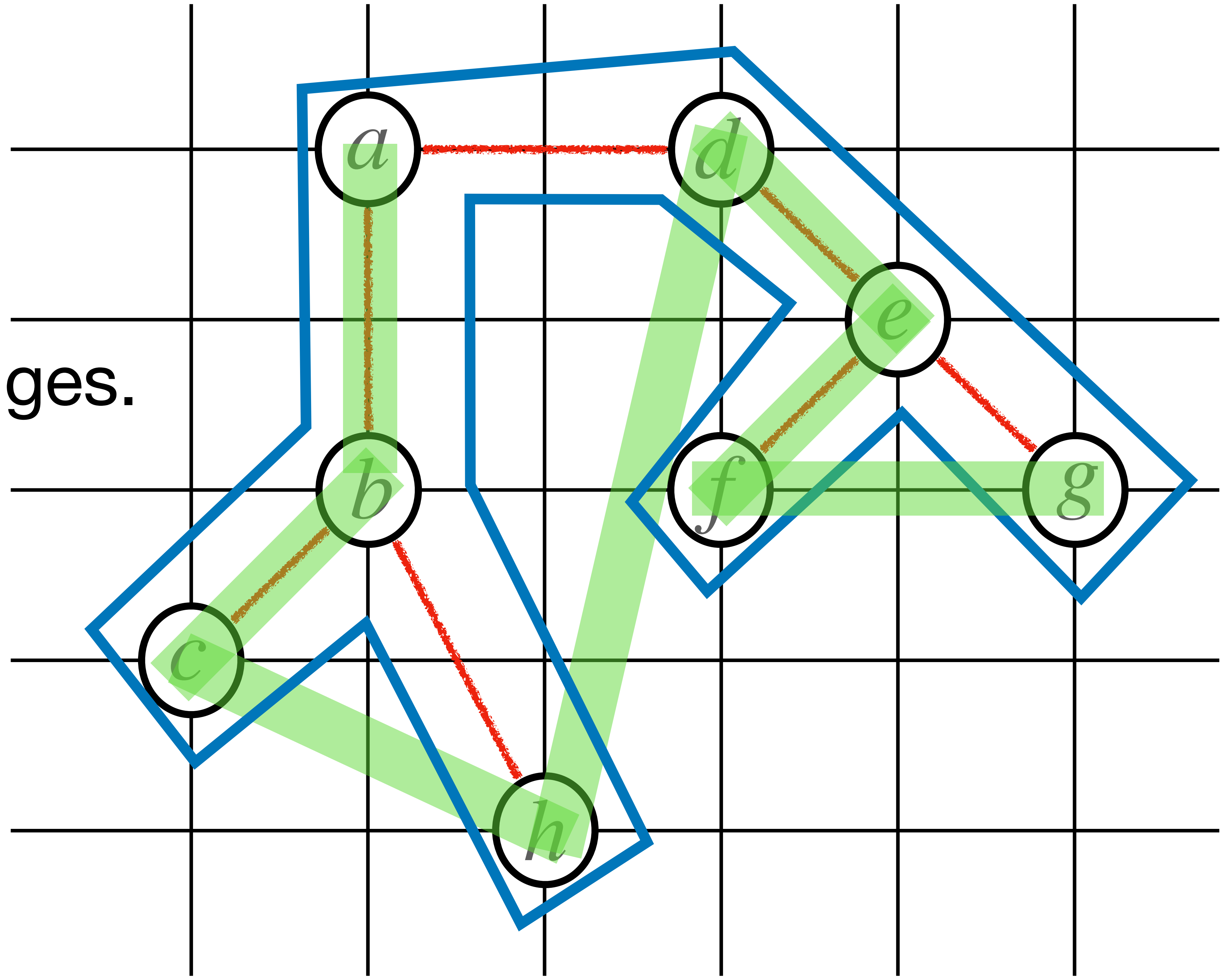
Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.

2. Get a cycle from the MST by taking a “tour" along its edges.

3. Get a Hamiltonian cycle by taking “short cuts”.



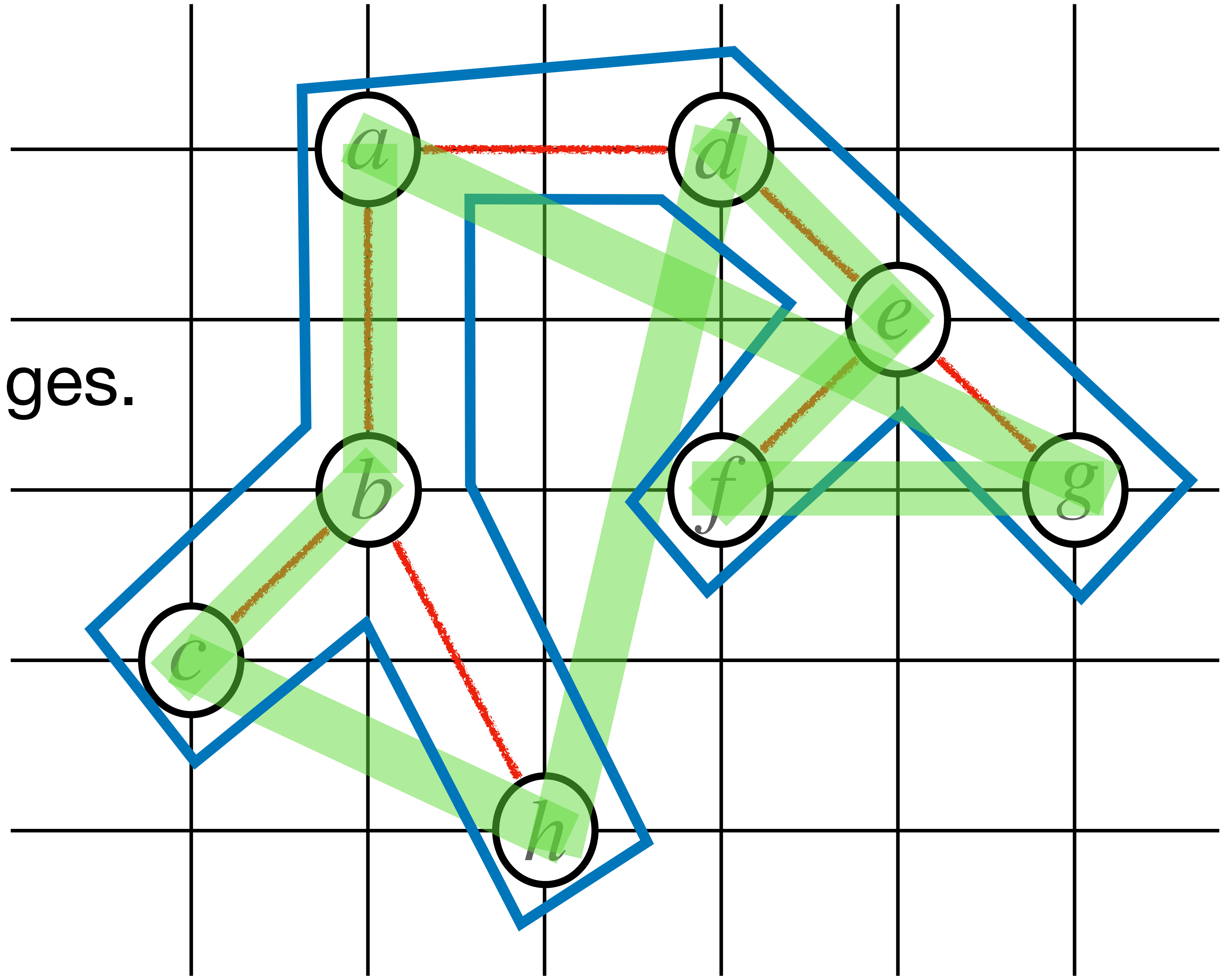
TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$. The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a “tour" along its edges.
3. Get a Hamiltonian cycle by taking “short cuts”.



TSP with Triangle Inequality

Input: An undirected complete graph G=(V,E), where every edge $(u, v) \in E$ has a non-negative integer weight $w(u, v)$.

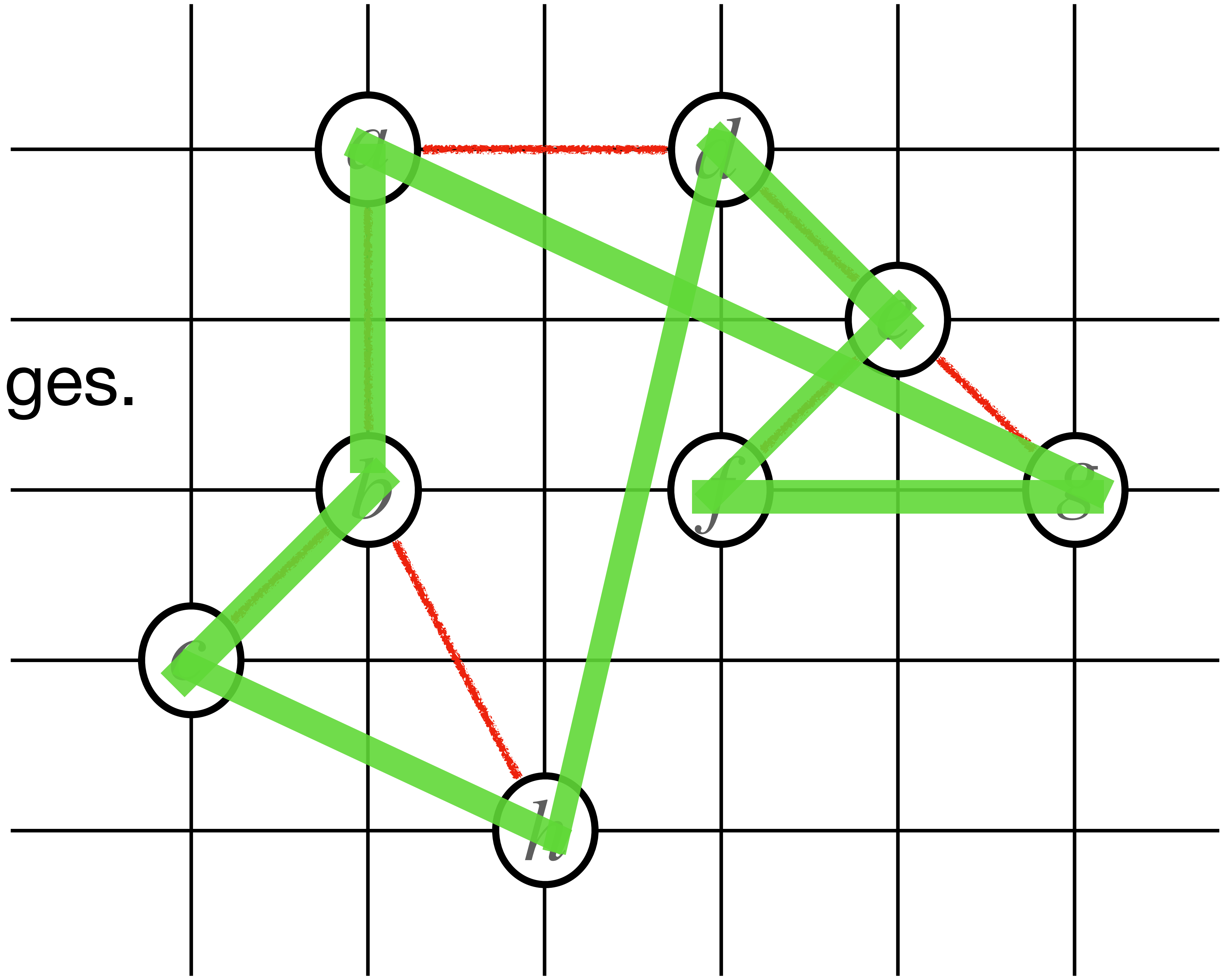
The edge weights satisfy the triangle inequality.

Output: A Hamiltonian cycle of minimum weight.

Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a “tour" along its edges.
3. Get a Hamiltonian cycle by taking “short cuts”.

See the Hamiltonian cycle more clearly.



Quiz questions:

1. What is the main idea of the above approximation algorithm for “TSP with triangle inequality”?

2. Can you think of an instance for which the above approximation algorithm will output an optimal solution, and an instance for which it will not?

Roadmap of this lecture:

1. Define "Approximation Algorithm".

2. Understand approximation algorithms by solving the "Vertex Cover Problem".

   2.1 An approximation algorithm for "Vertex Cover Problem".

   2.2 Analyze the approximation ratio of the algorithm.

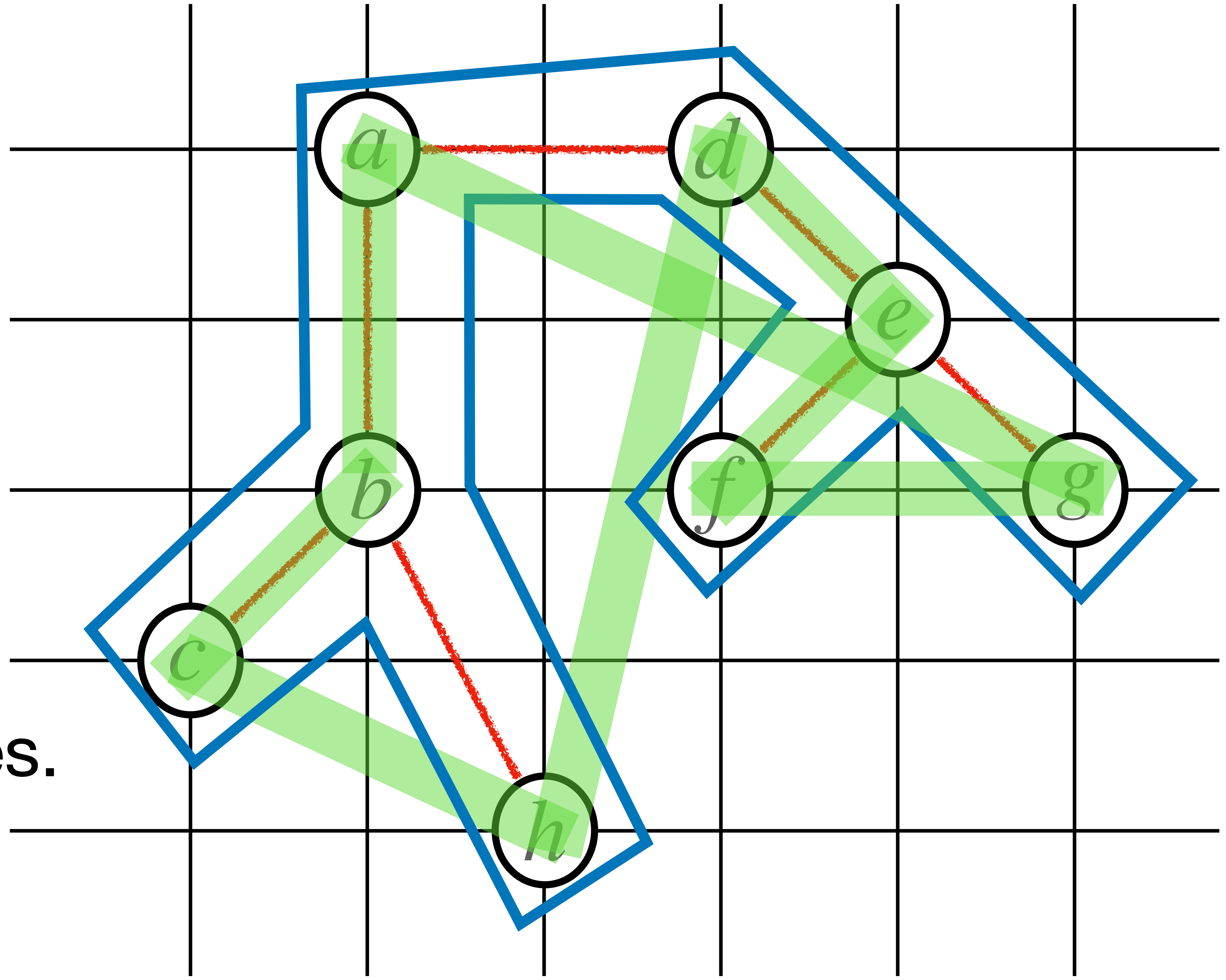
3. Understand approximation algorithms by solving the "Traveling Salesman Problem (TSP)".

   3.1 An approximation algorithm for TSP with the triangle inequality.

   3.2 Analyze the approximation ratio of the algorithm.

Theorem: The above algorithm for TSP with triangle inequality is a polynomial-time 2-approximation algorithm.

Proof:



Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a “tour" along its edges.
3. Get a Hamiltonian cycle by taking “short cuts”.

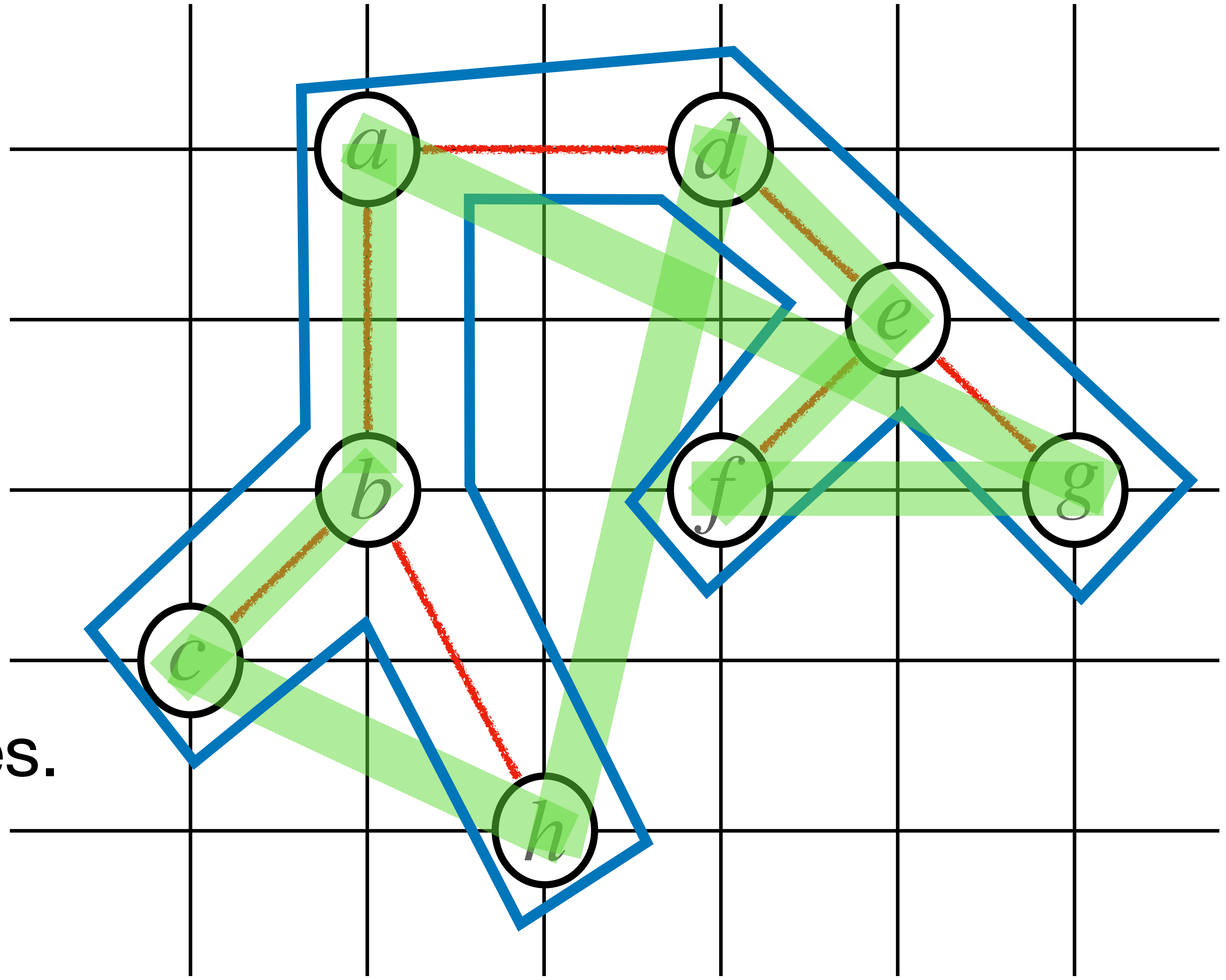
Theorem: The above algorithm for TSP with triangle inequality is a polynomial-time 2-approximation algorithm.

Proof: C_{MST} : Weight of the MST

C_{DFS} : Weight of the "tour" obtained in Step 2.

C : Weight of the Hamiltonian cycle we found in Step 3.

C^* : Minimum weight of an optimal Hamiltonian cycle.



Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a "tour" along its edges.
3. Get a Hamiltonian cycle by taking "short cuts".

Theorem: The above algorithm for TSP with triangle inequality is a polynomial-time 2-approximation algorithm.

Proof: C_{MST} : Weight of the MST

C_{DFS} : Weight of the "tour" obtained in Step 2.

C : Weight of the Hamiltonian cycle we found in Step 3.

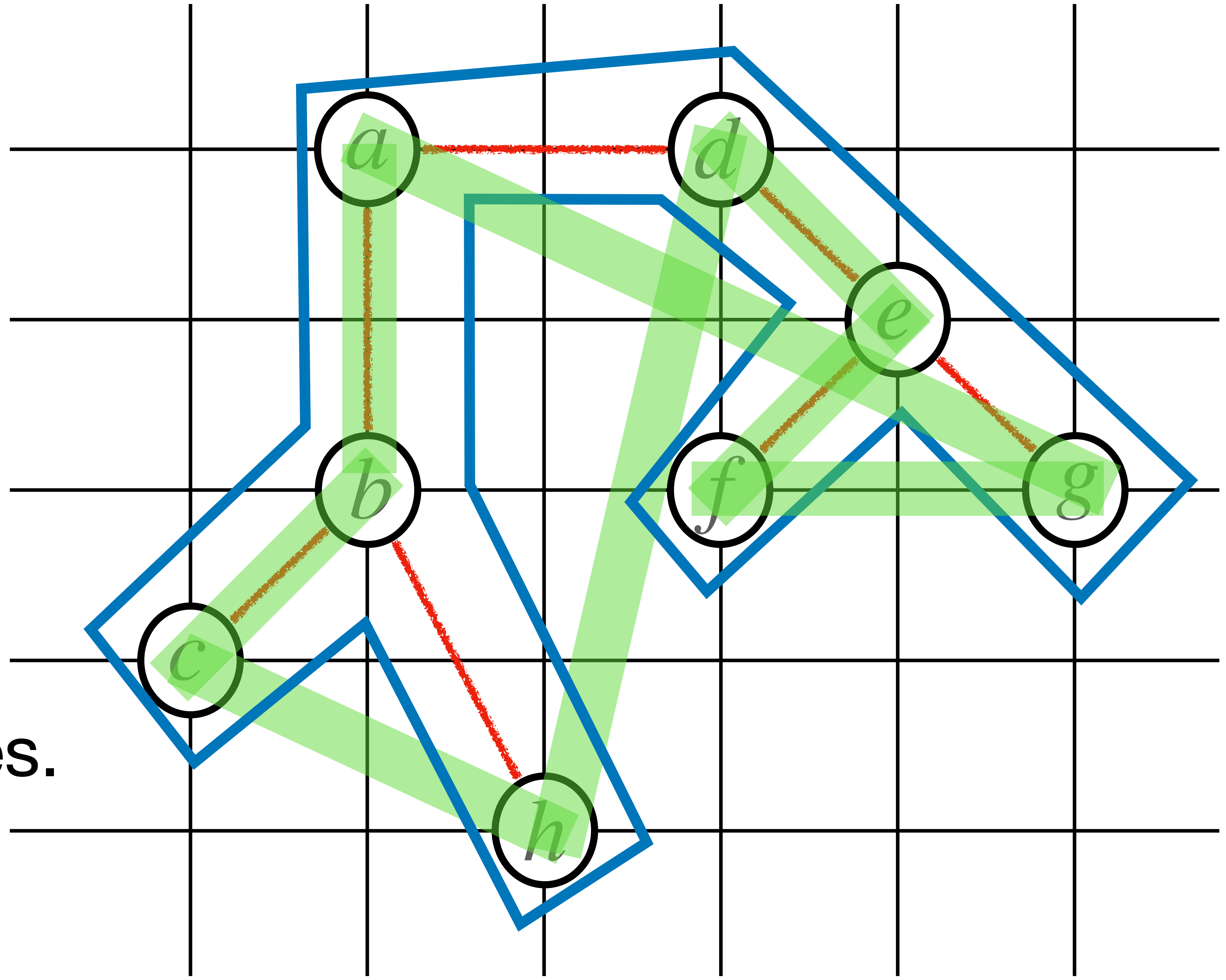
C^* : Minimum weight of an optimal Hamiltonian cycle.

$\boxed{C_{MST} \leq C^*}$ Why?

Lower bound to C^*

Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a "tour" along its edges.
3. Get a Hamiltonian cycle by taking "short cuts".



Theorem: The above algorithm for TSP with triangle inequality is a polynomial-time 2-approximation algorithm.

Proof: C_{MST} : Weight of the MST

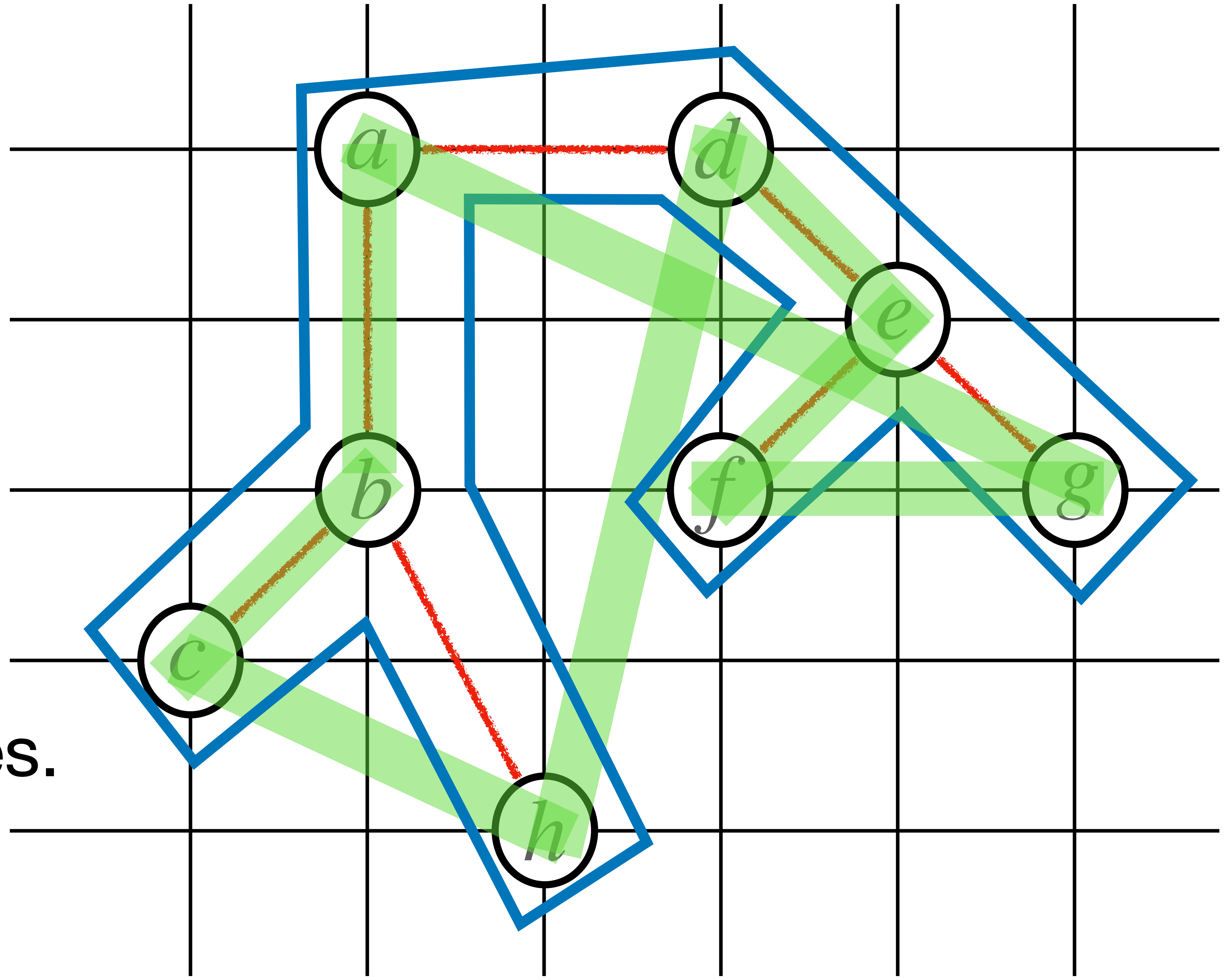
C_{DFS} : Weight of the "tour" obtained in Step 2.

C : Weight of the Hamiltonian cycle we found in Step 3.

C^* : Minimum weight of an optimal Hamiltonian cycle.

$\boxed{C_{MST} \leq C^*}$ $\boxed{C_{DFS} = 2C_{MST}}$

Why?



Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a "tour" along its edges.
3. Get a Hamiltonian cycle by taking "short cuts".

Theorem: The above algorithm for TSP with triangle inequality is a polynomial-time 2-approximation algorithm.

Proof: C_{MST} : Weight of the MST

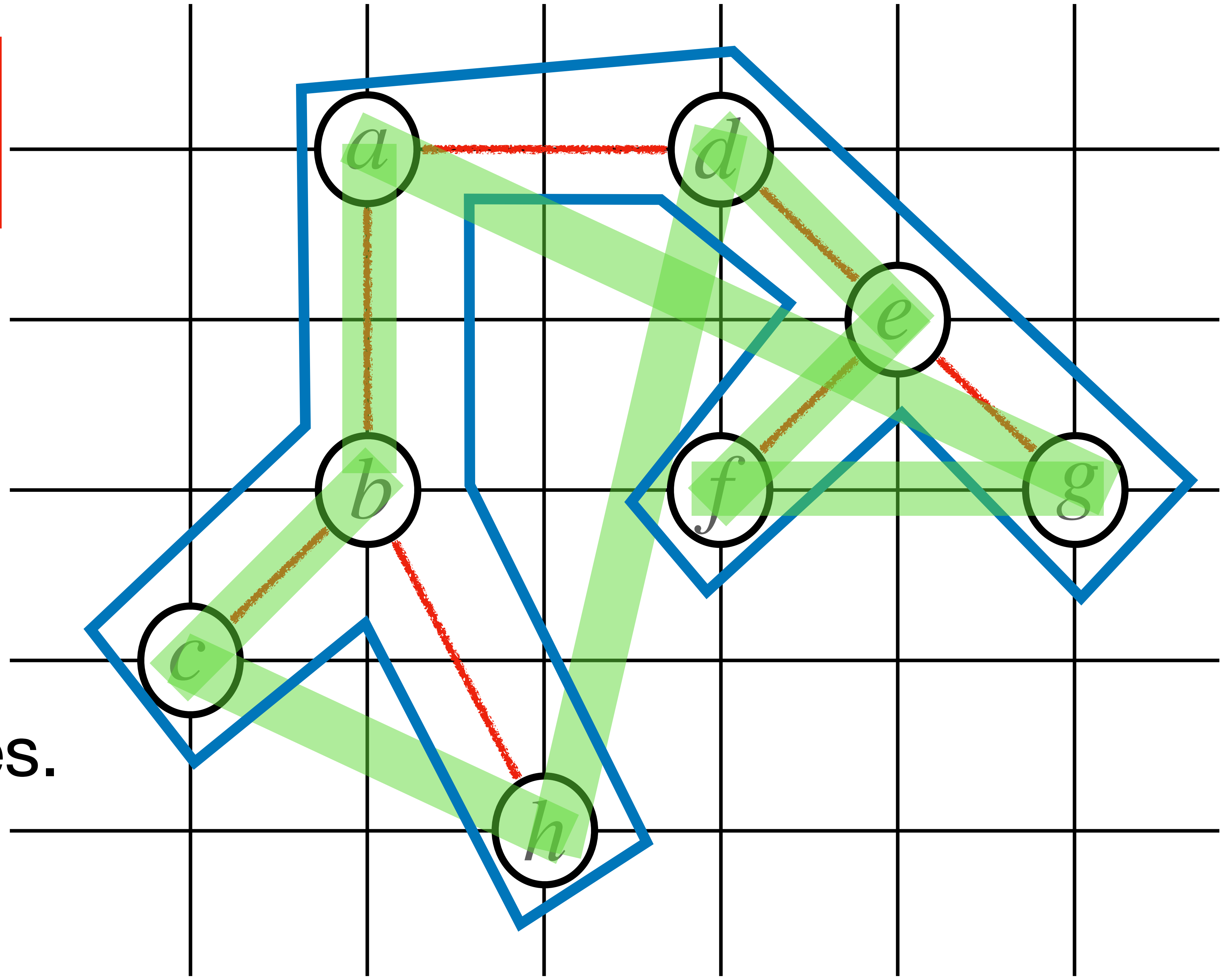
C_{DFS} : Weight of the "tour" obtained in Step 2.

C : Weight of the Hamiltonian cycle we found in Step 3.

C^* : Minimum weight of an optimal Hamiltonian cycle.

$\boxed{C_{MST} \leq C^*}$ $\boxed{C_{DFS} = 2C_{MST}}$ $\boxed{C \leq C_{DFS}}$

Why?



Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a "tour" along its edges.
3. Get a Hamiltonian cycle by taking "short cuts".

Theorem: The above algorithm for TSP with triangle inequality is a polynomial-time 2-approximation algorithm.

Proof: C_{MST} : Weight of the MST

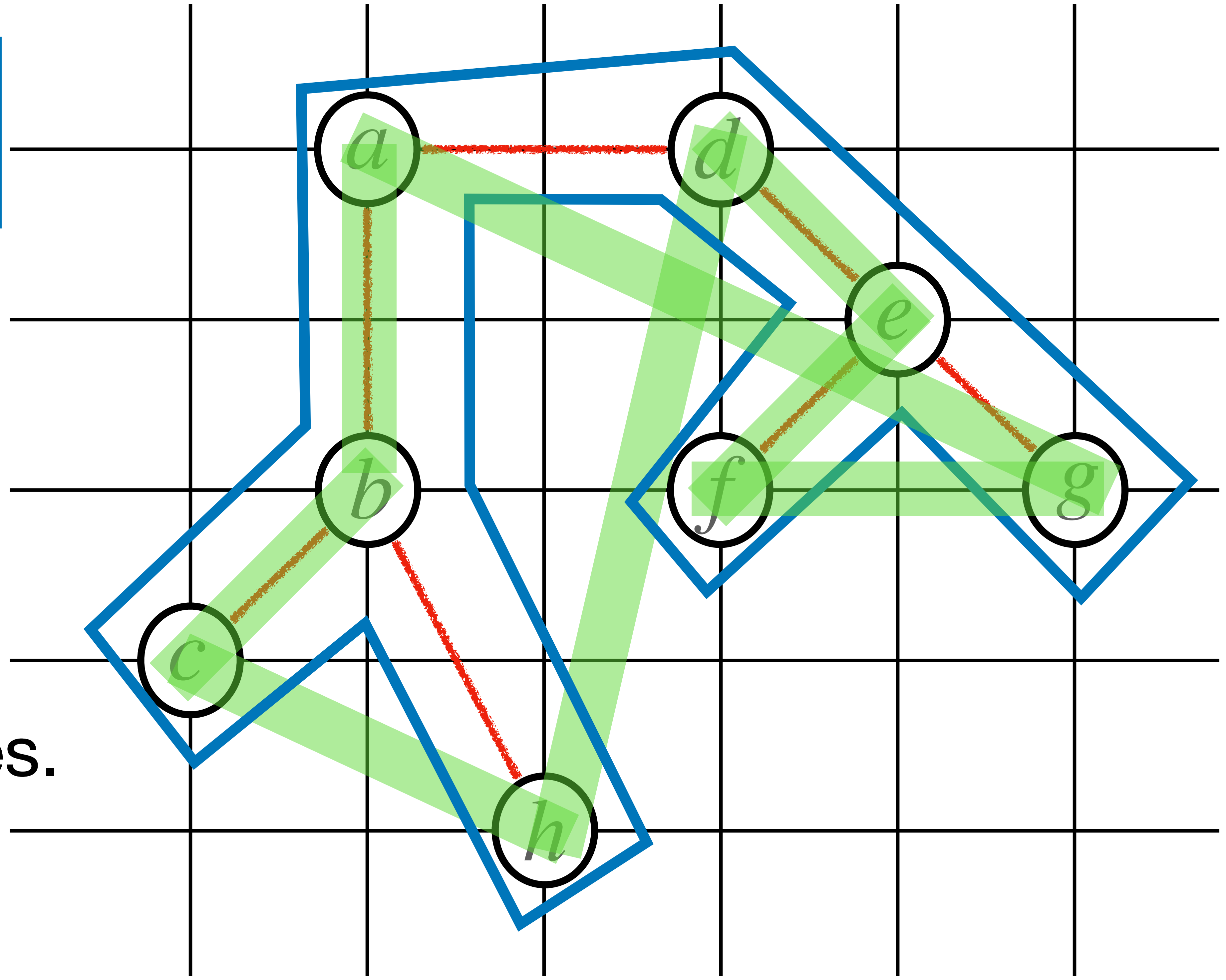
C_{DFS} : Weight of the "tour" obtained in Step 2.

C : Weight of the Hamiltonian cycle we found in Step 3.

C^* : Minimum weight of an optimal Hamiltonian cycle.

$\boxed{C_{MST} \leq C^*}$ $\boxed{C_{DFS} = 2C_{MST}}$ $\boxed{C \leq C_{DFS}}$

$$\boxed{C \leq C_{DFS} = 2C_{MST} \leq 2C^*}$$



Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a "tour" along its edges.
3. Get a Hamiltonian cycle by taking "short cuts".

Theorem: The above algorithm for TSP with triangle inequality is a polynomial-time 2-approximation algorithm.

Proof: C_{MST} : Weight of the MST

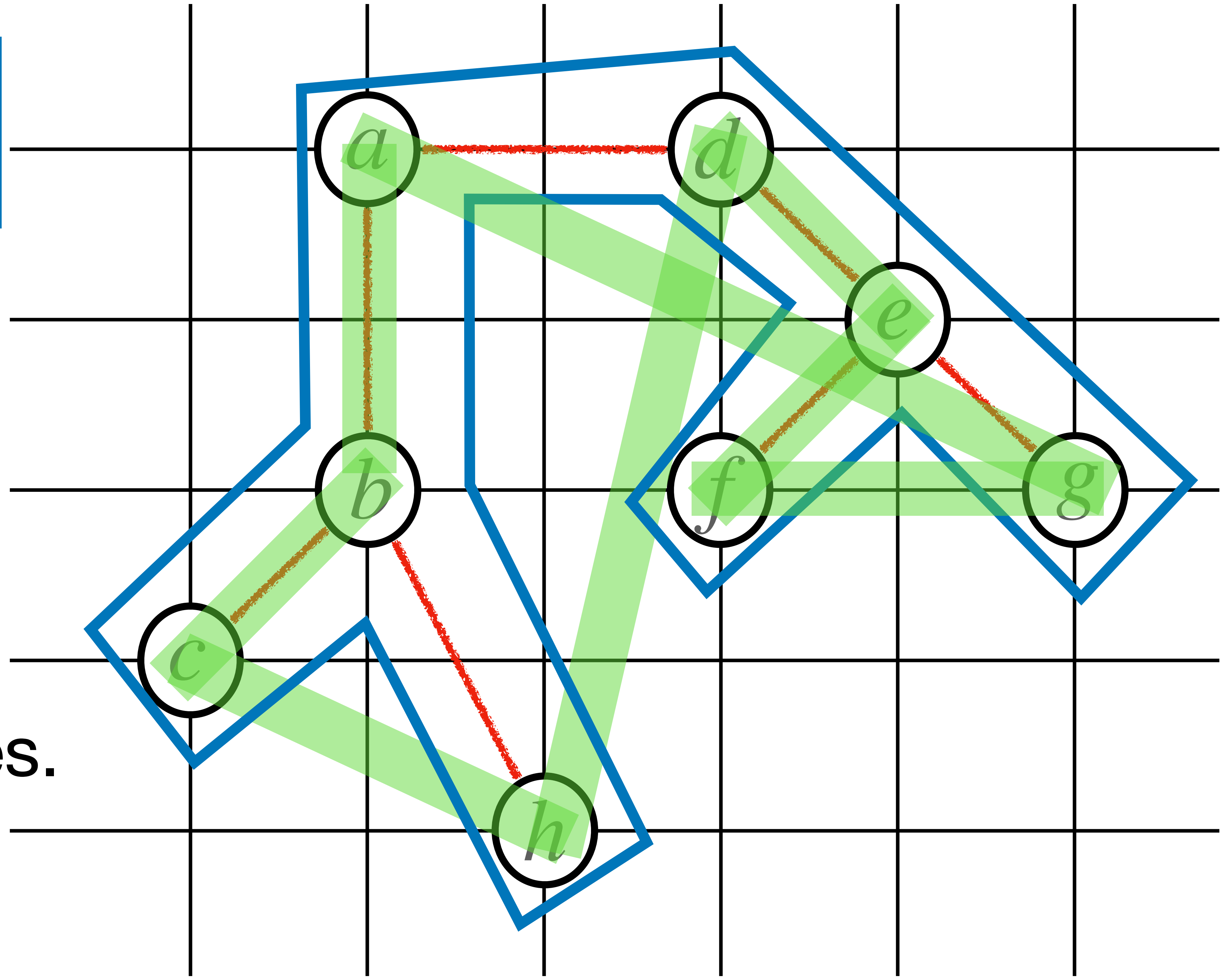
C_{DFS} : Weight of the "tour" obtained in Step 2.

C : Weight of the Hamiltonian cycle we found in Step 3.

C^* : Minimum weight of an optimal Hamiltonian cycle.

$\boxed{C_{MST} \leq C^*}$ $\boxed{C_{DFS} = 2C_{MST}}$ $\boxed{C \leq C_{DFS}}$

$$\boxed{C \leq C_{DFS} = 2C_{MST} \leq 2C^*} \Rightarrow \frac{C}{C^*} \leq 2$$



Idea of Algorithm:

1. Build an MST of G.
2. Get a cycle from the MST by taking a "tour" along its edges.
3. Get a Hamiltonian cycle by taking "short cuts".

Quiz questions:

1. For the above approximation algorithm for "TSP with triangle inequality", how did we find its approximation ratio without knowing the optimal cost?

2. If we do not have the "triangle inequality condition" for the TSP, will the above proof for the approximation ratio still work?