APPROXIMATION ALGORITHMS

ADVANCED ALGORITHMS

MARK FLORYAN

CLRS CH. 35

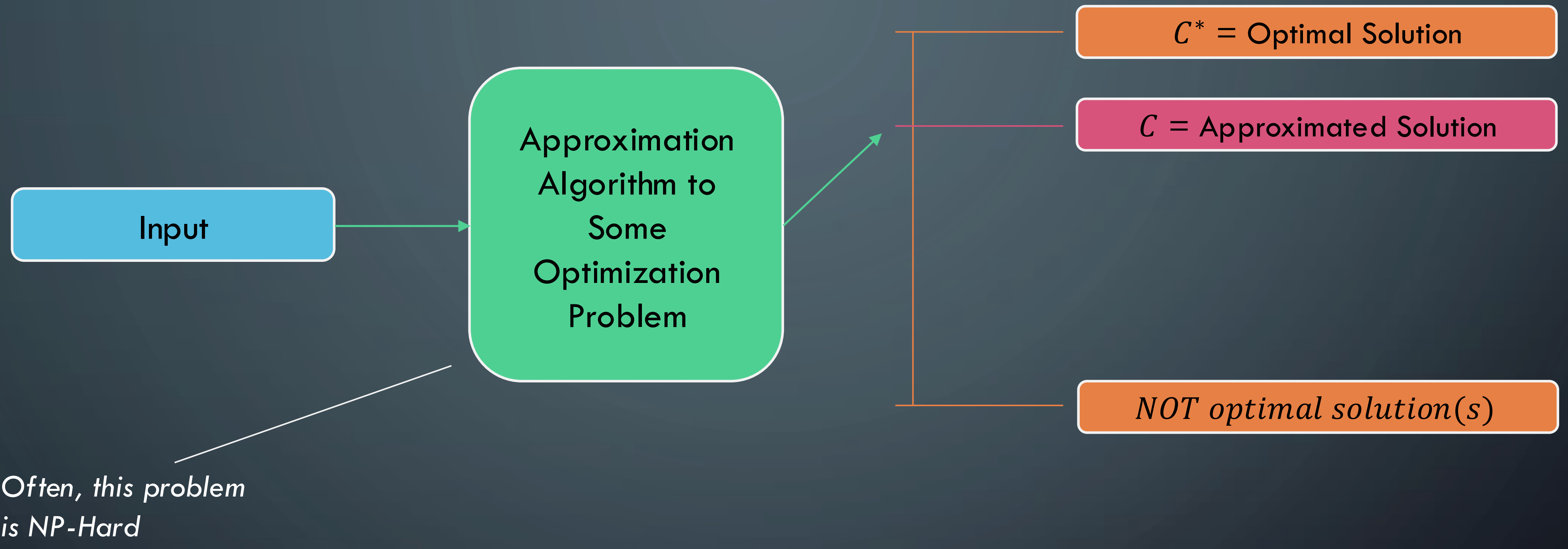
ADVANCED TREE STRUCTURES

In this deck we will look at:

- ***Approximation Ratios***
- ***Vertex Cover Approximation***
- ***TSP Approximation***
- ***Set-Covering Approximation***

INTRODUCTION: APPROXIMATION

APPROXIMATION RATIOS



An approximation algorithm has an ***approximation ratio*** of $\rho(n)$ if, for any input size n, the cost C produced by the algorithm satisfies:

$$Max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

SOME DEFINITIONS

$\rho(n)$-Approximation Algorithm	*An approximation algorithm that achieves an approximation of $\rho(n)$ on all inputs*
Approximation Scheme	*An approximation algorithm that takes as input the instance of the problem and a value $\epsilon > 0$ such that for any fixed ϵ, the scheme is a $(1 + \epsilon)$-approximation algorithm.*
Polynomial Time Approximation Scheme	*A scheme is polynomial time if, for any fixed $\epsilon > 0$, the algorithm runs in polynomial time in terms of the input size n*
Fully Polynomial Time Approximation Scheme	*A scheme is fully polynomial time if it also has a runtime that is polynomial in terms of $\frac{1}{\epsilon}$.*

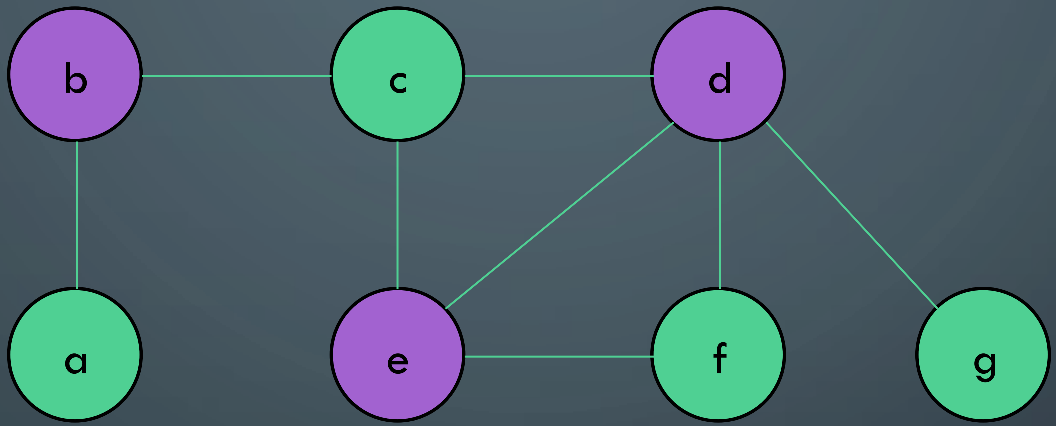
For example, a fully polynomial time approximation scheme might have runtime $\Theta\left(\left(\frac{1}{\epsilon}\right)^2 n^2\right)$

VERTEX-COVER

RECALL VERTEX-COVER

Given a Graph $G = (V, E)$, find the smallest set of nodes $V' \subset V \mid \forall_{e=(u,v)\in E} \, (u \in V' \vee v \in V')$

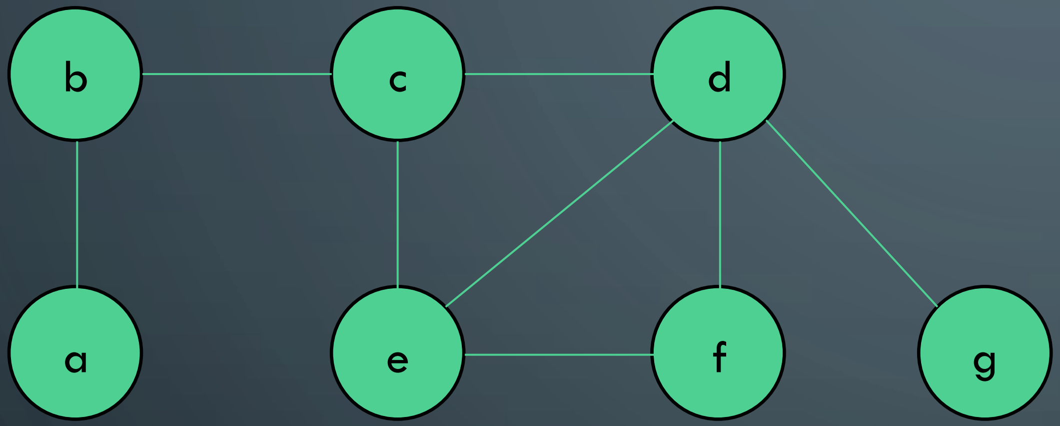
This problem is NP-Hard



In other words, find the smallest set of nodes that touches every edge at least once

The purple nodes are the smallest solution for this graph

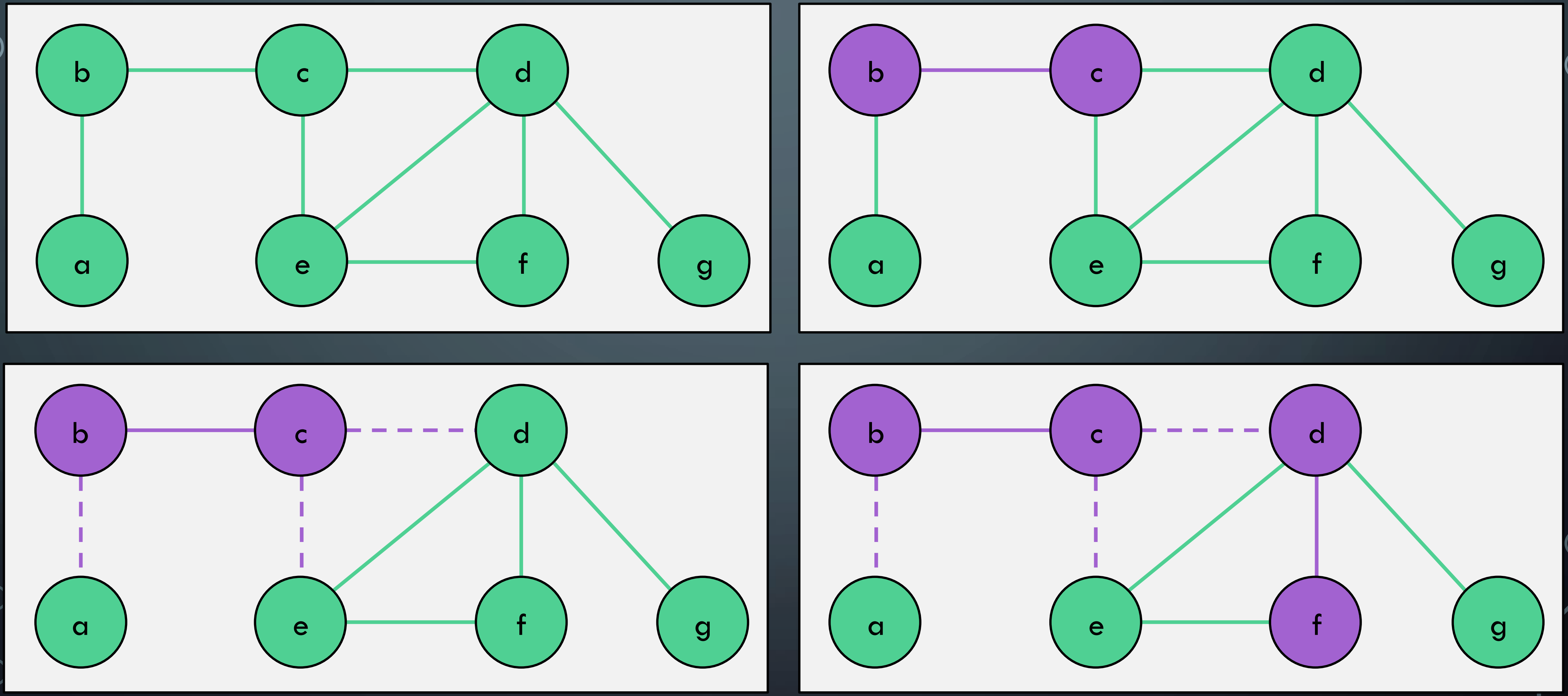
PROPOSED ALGORITHM FOR VERTEX-COVER




```
Vertex-Approximation(G):
  C = []
  E' = G.E
  while E' not empty:
    let e=(u,v) be arbitrary edge from E'
    C = C.append({u,v})
    remove all edges in E' incident on u or v
  return C
```

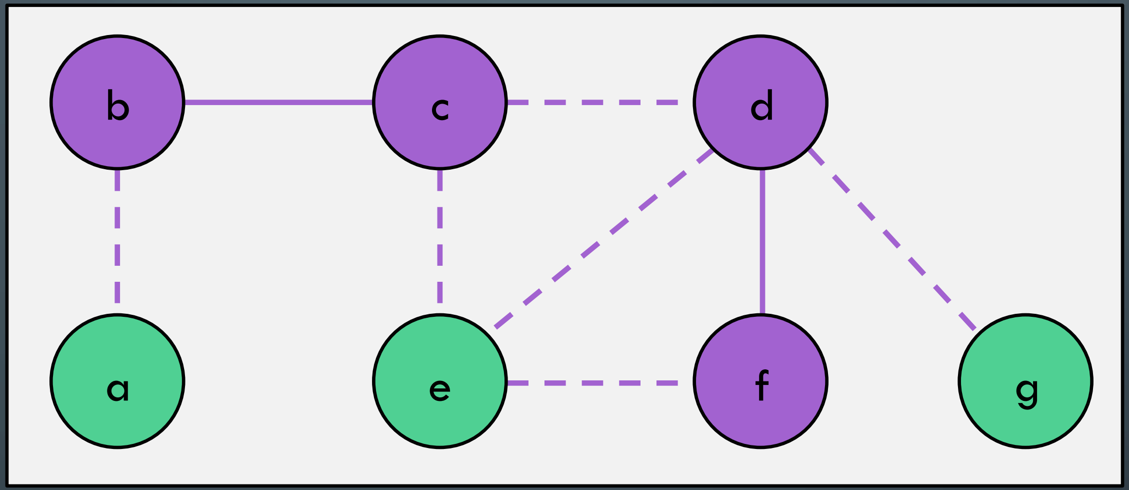
Run this on the given graph a few times. What is the approximation ratio? Do we have a guaranteed ratio here?

SAMPLE EXECUTION

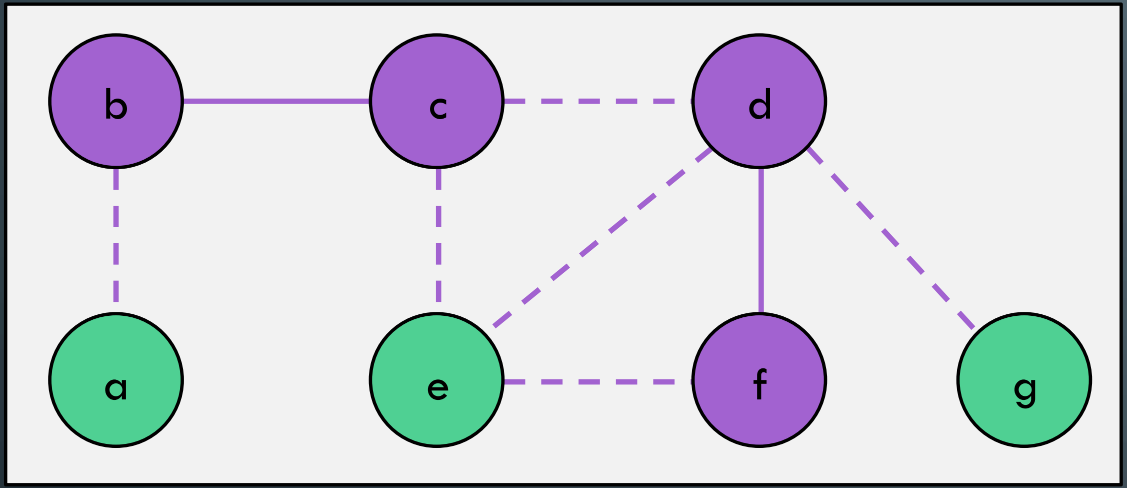


...last step is on next slide

SAMPLE EXECUTION



SAMPLE EXECUTION




```
Vertex-Approximation(G):
  C = []
  E' = G.E
  while E' not empty:
    let e=(u,v) be arbitrary edge from E'
    C = C.append({u,v})
    remove all edges in E' incident on u or v
  return C
```

This is just one way it could have turned out. Any thoughts on the limits of "how bad" or "how good" the approximation will be?

APPROX-VERT-COVER: ANALYSIS

Runtime is polynomial? Yes, it is $O(|E|^2)$

```
Vertex-Approximation(G):
  C = []
  E' = G.E
  while E' not empty:
    let e=(u,v) be arbitrary edge from E'
    C = C.append(e)
    remove all edges in E' incident on u or v
  return C
```

For each edge e we choose on this line, the solution definitely includes u or v, we always add both

We remove as many extra edges as we can here, but of course there could be 0 each time.

APPROX-VERT-COVER: ANALYSIS

Claim: Vertex-Approximation is a 2-approximation of vertex cover

```
Vertex-Approximation(G):
  C = []
  E' = G.E
  while E' not empty:
    let e=(u,v) be arbitrary edge from E'
    C = C.append({u,v})
    remove all edges in E' incident on u or v
  return C
```

Proof Part 1 (Correctness): The nodes returned by this algorithm are a valid vertex cover

This is self-evident. Every edge in E is either chosen in line 4 and both it's nodes added to the solution OR removed in line 6 being incident to u or v.

Thus, every edge is incident on some node in the set C because we do not terminate until E' is empty.

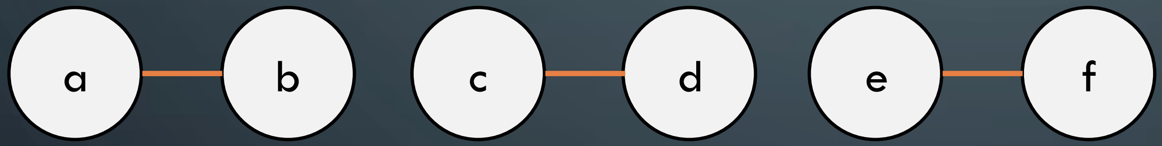
The algorithm returns a valid vertex-cover

APPROX-VERT-COVER: ANALYSIS

Claim: Vertex-Approximation is a 2-approximation of vertex cover

Proof Part 2 (Ratio): The size of C is no more than twice the size of the optimal vertex-cover

Consider the set of edges A chosen in line 4 of the algorithm. Because other incident edges are removed in line 6, the lines in A form a set of disjoint pairs



There may be edges between these pairs (not shown), but the vertex cover MUST include at least one node from each pair. Thus, we get the inequality $|C^*| \geq |A|$

Separately, note that our vertex cover returned always returns exactly two nodes for each edge in A, thus:

$$|C| = 2|A|$$

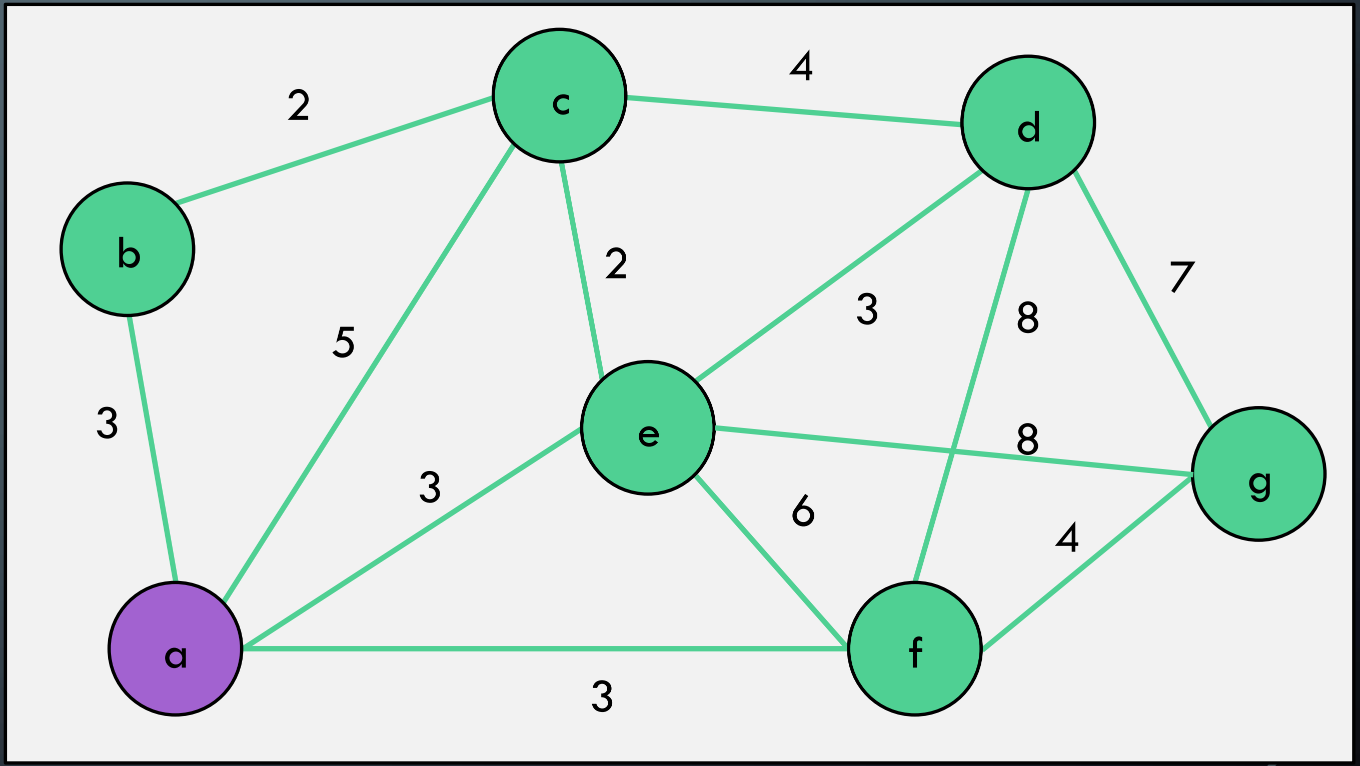
Combining these equations we get:

$$|C| = 2|A|$$
$$|C| \leq 2|C^*|$$

TRAVELING SALESPERSON

RECALL TRAVELING SALESPERSON PROBLEM

Recall the traveling salesperson problem: Given a graph G and start node, what is the minimum cost tour (visit every node exactly once then return to start)

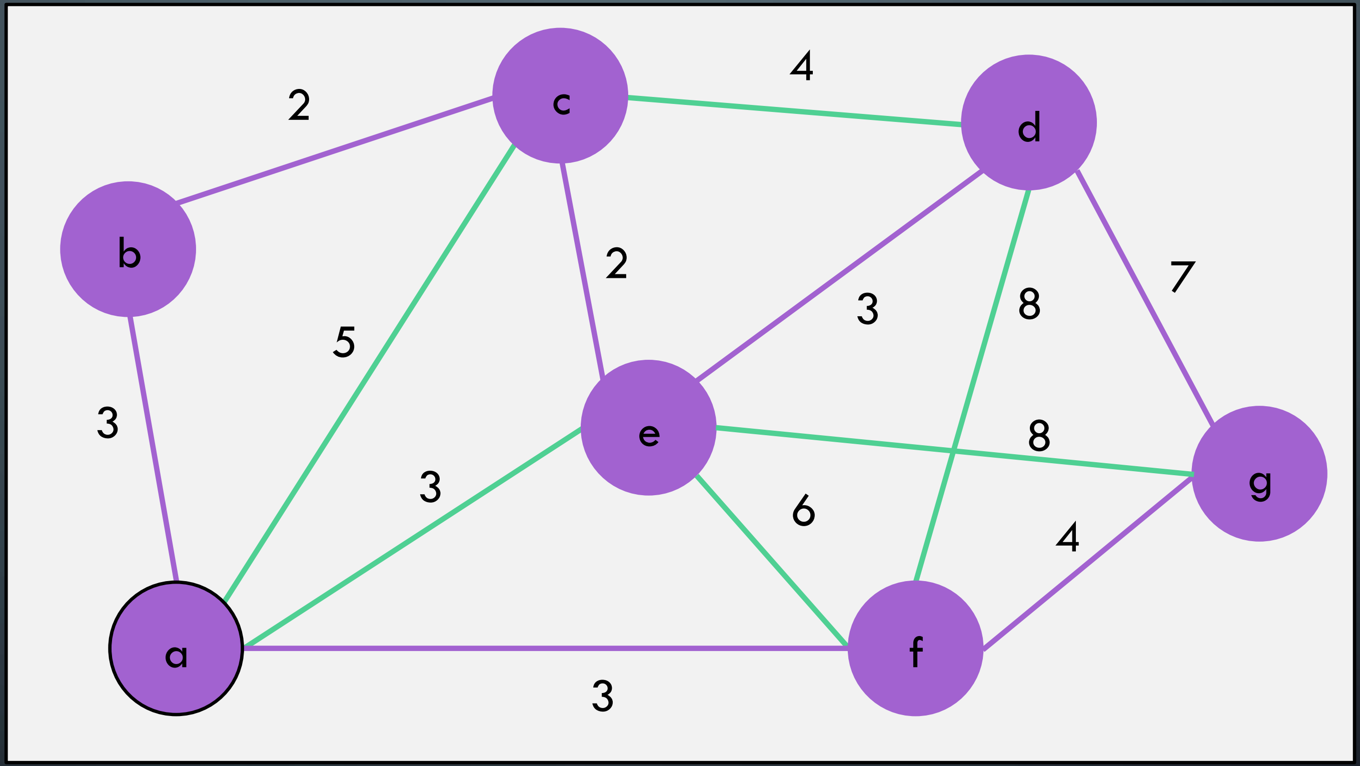


For our approximation, we are going to assume the graph is COMPLETE and that the triangle inequality holds. For any three nodes u, v, w:

$$c(u,w) \leq c(u,v) + c(v,w)$$

This graph is not complete but is sufficient for showcasing out point

RECALL TRAVELING SALESPERSON PROBLEM



Total Cost of this tour is: 3 + 2 + 2 + 3 + 7 + 4 + 3 = 24

APPROXIMATION FOR TSP

```
TSP-Approx(G, s):
   Select any vertex r in G.V
   MST T = prims-mst(G, r)
   prev = T.preorder().first
   for v in T.preorder().second : …
      add (prev, v) to solution
   add (prev, s) to solution
   return solution
```

Basic Idea:

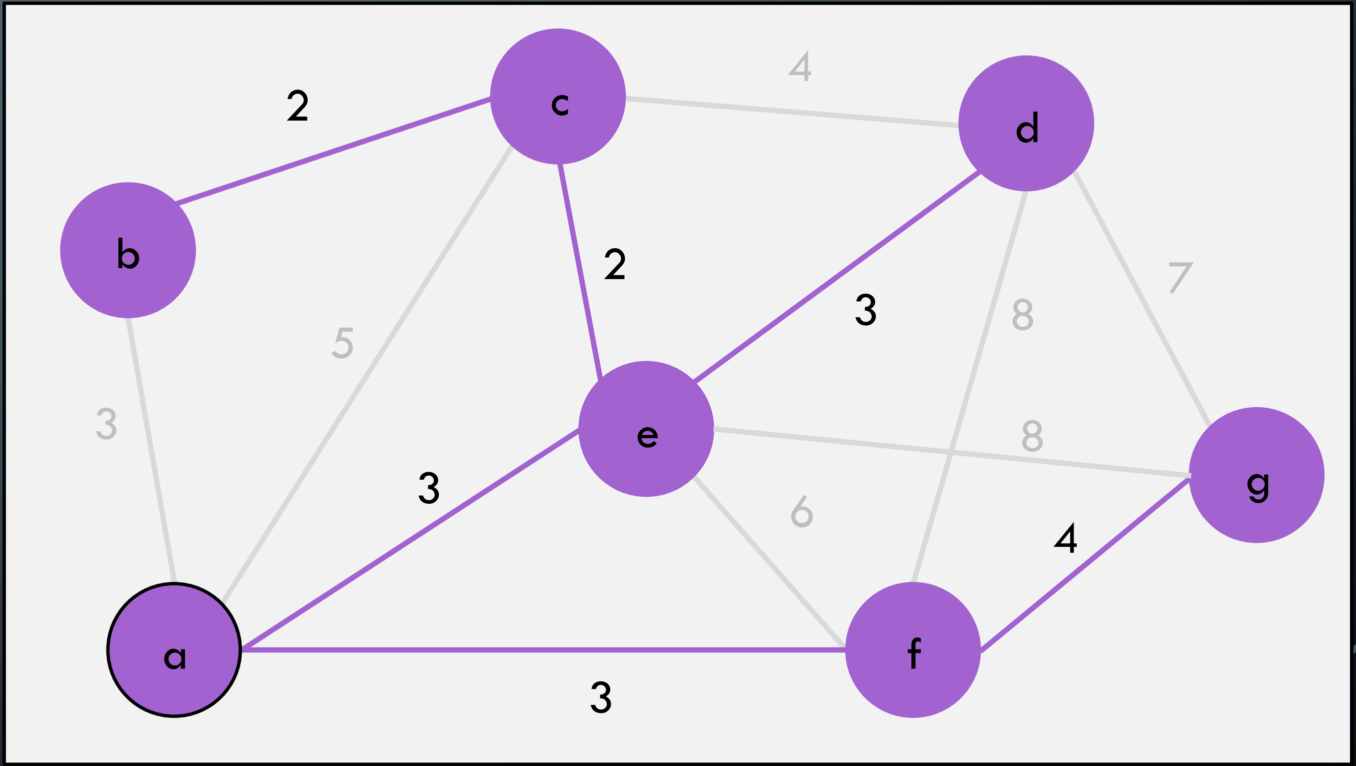
Calculate minimum spanning tree of G. Use pre-order traversal of this MST to define the tour for TSP

APPROXIMATION FOR TSP

Step 1:

Calculate MST of G (let's root it at a)

```
TSP-Approx(G, s):
   Select any vertex R in G.V
   MST T = prims-mst(G, r)
   prev = T.preorder().first
   for v in T.preorder().second : …
      add (prev, v) to solution
   add (prev, s) to solution
   return solution
```

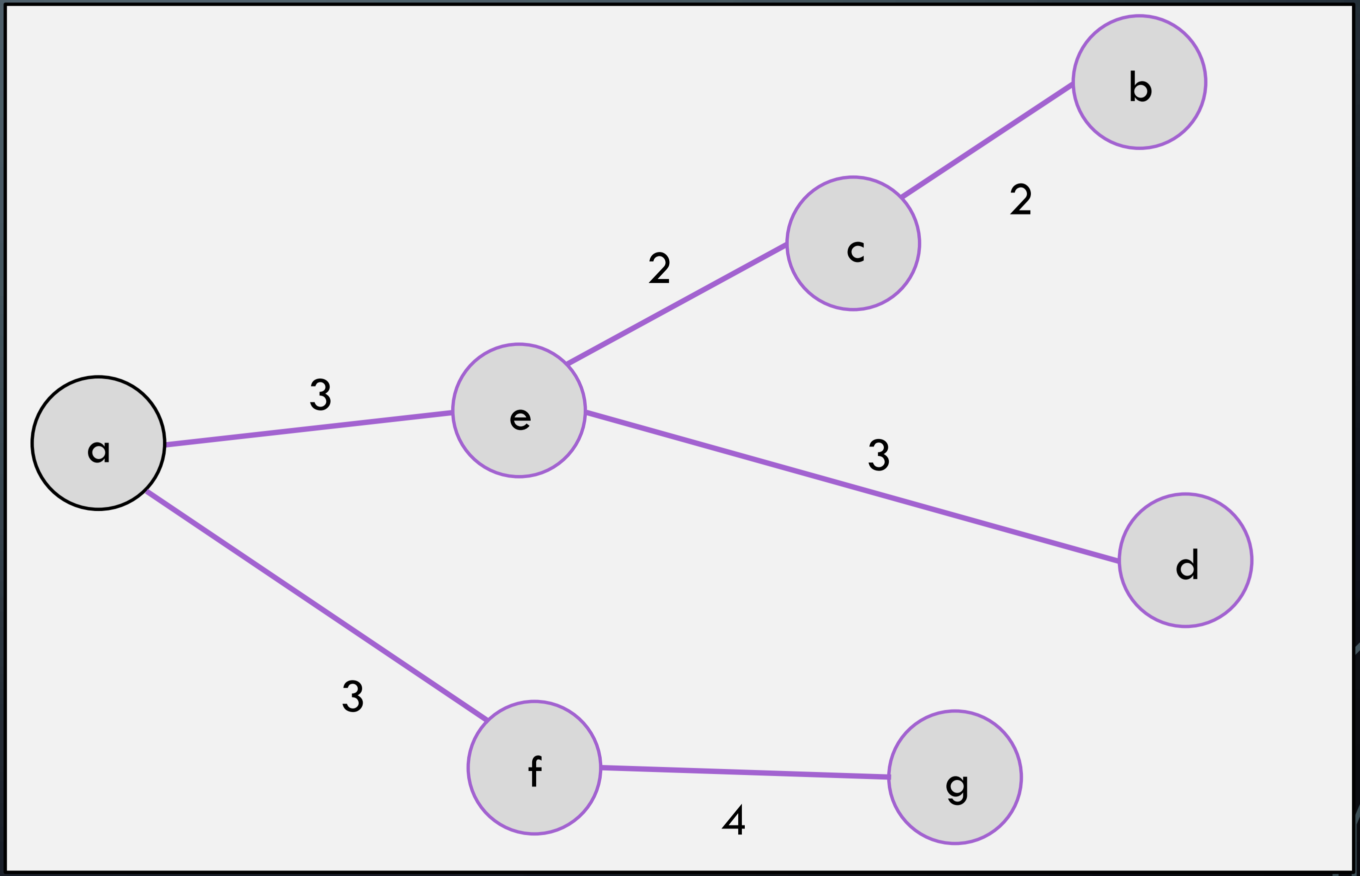


APPROXIMATION FOR TSP

Step 1.5:

Re-arrange the MST to make it easier to view, remove unused edges

```
TSP-Approx(G, s):
   Select any vertex R in G.V
   MST T = prims-mst(G, r)
   prev = T.preorder().first
   for v in T.preorder().second : …
      add (prev, v) to solution
   add (prev, s) to solution
   return solution
```

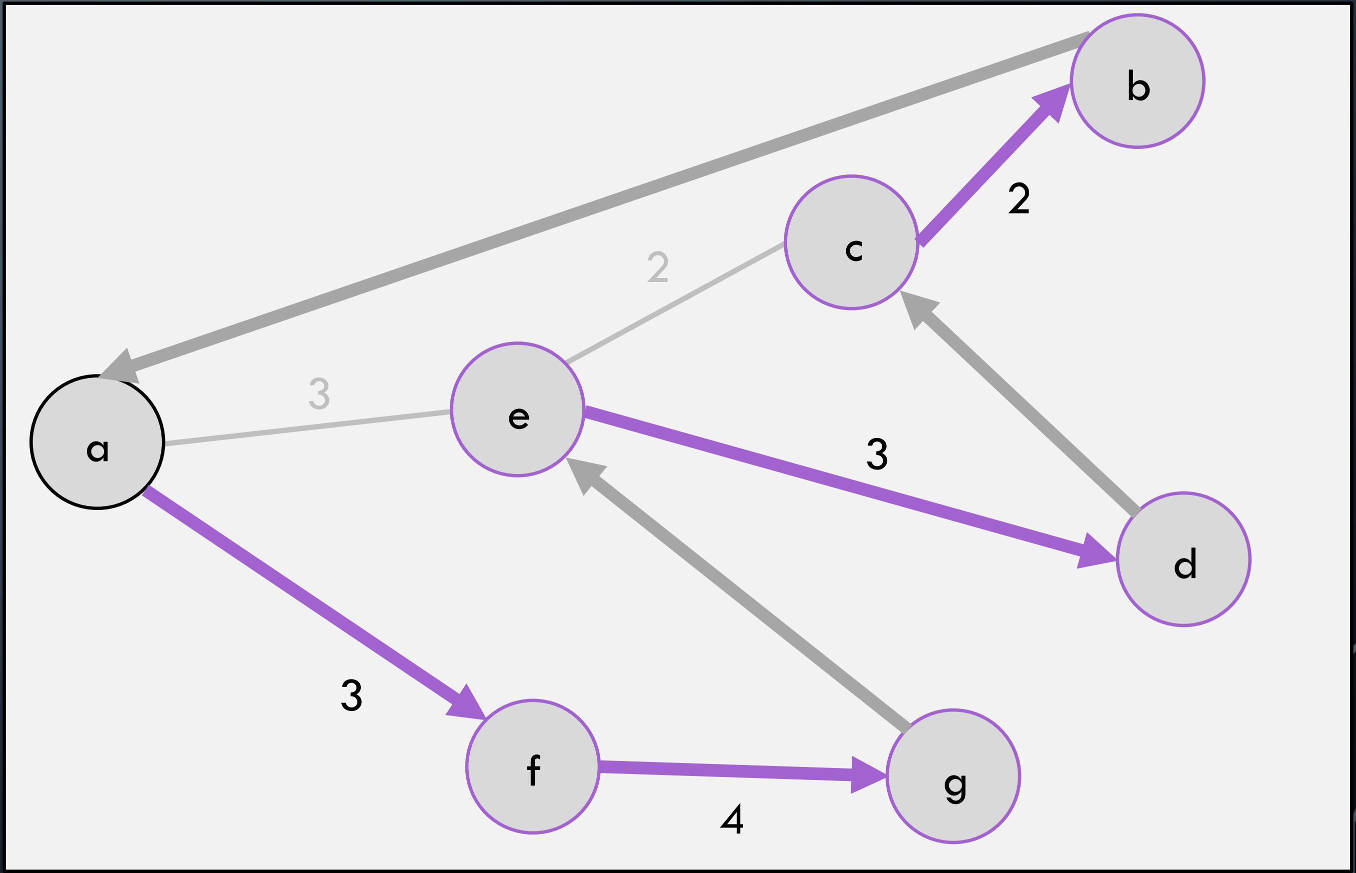


APPROXIMATION FOR TSP

```
TSP-Approx(G, s):
   Select any vertex R in G.V
   MST T = prims-mst(G, r)
   prev = T.preorder().first
   for v in T.preorder().second : …
      add (prev, v) to solution
   add (prev, s) to solution
   return solution
```

Step 2:

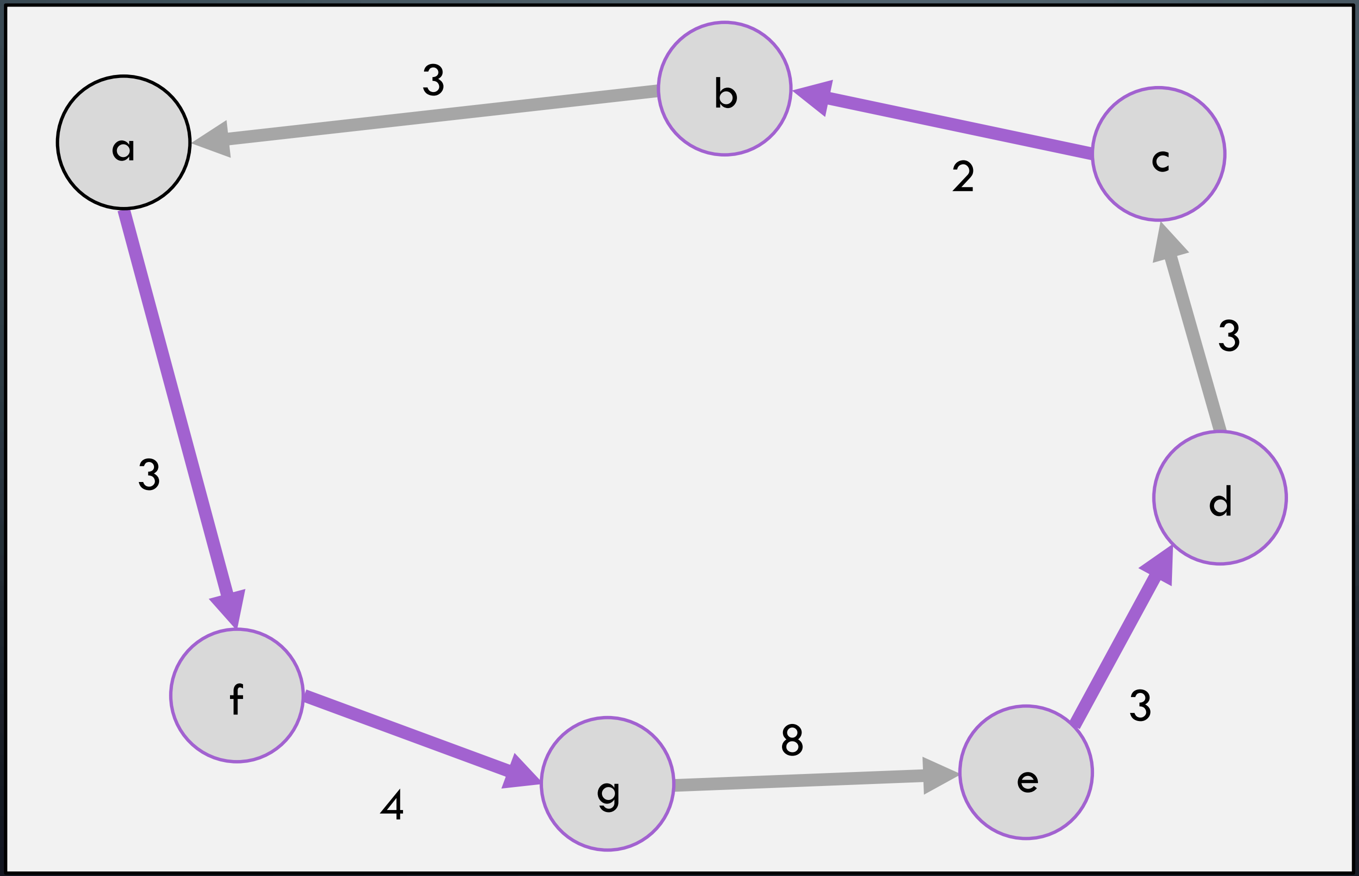
Tour is just the pre-order traversal of the MST. Note that graph needs to be complete to ensure gray edges definitely exist



APPROXIMATION FOR TSP

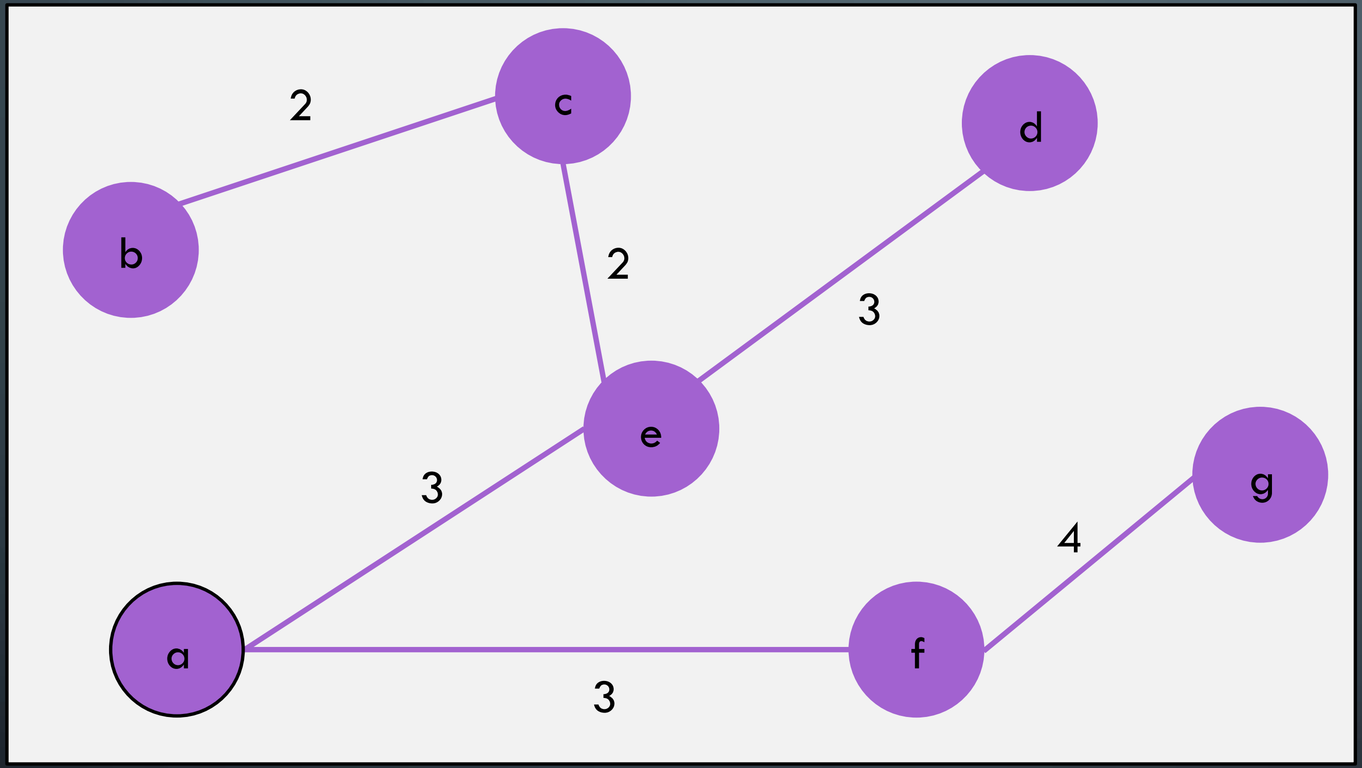
Rearrange to show tour:

Cost is 3 + 4 + 8 + 3 + 3 + 2 + 3 = 26



How good / bad can this approximation be though?

APPROXIMATION FOR TSP: ANALYSIS



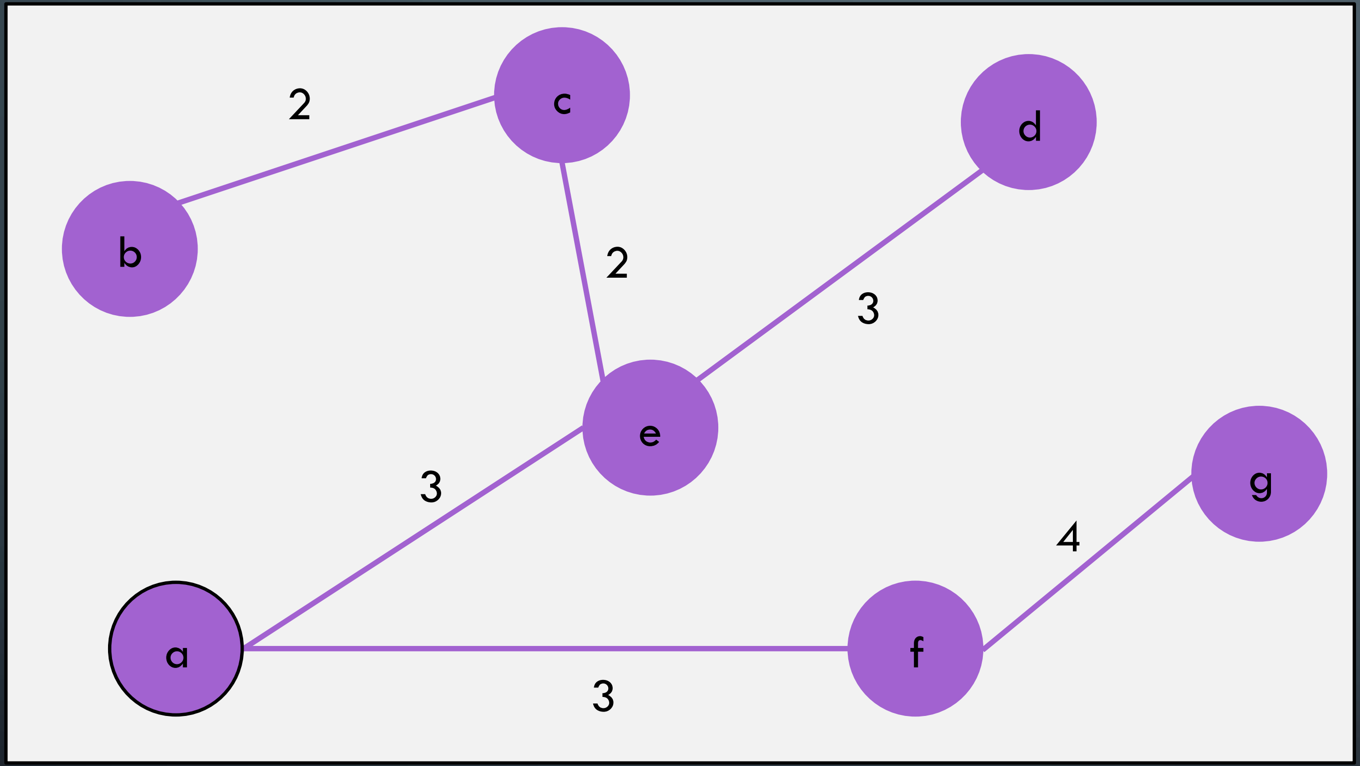
So we get the inequality:

$$c(T) \leq c(H^*)$$

Observation 1:

The minimum spanning tree is a lower bound for the optimal tour.
Why?

APPROXIMATION FOR TSP: ANALYSIS



***So we get the inequality*:**

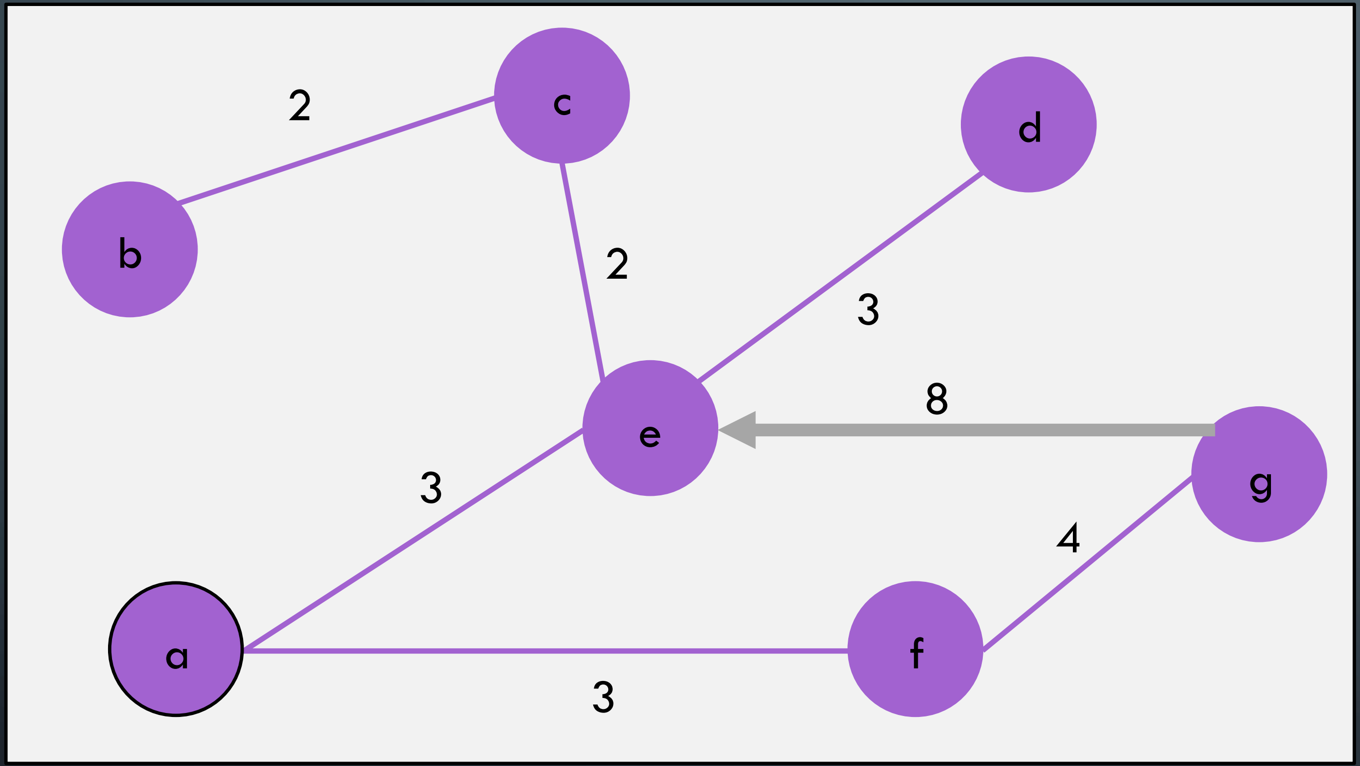
$$c(W) = 2c(T)$$
$$2c(T) \leq 2c(H^*)$$

But this walk is not a tour (it is an mst full walk), what about the tour that gets returned by the algorithm?

***Observation 2*:**

Consider the FULL WALK of T. Notice that the full walk traverses each edge exactly twice.
Full walk is (a f g f a e d e c b c e a)

APPROXIMATION FOR TSP: ANALYSIS



Observation 3:

The cross edges never increase the cost of the "walk"

Consider the edge (g,e) with cost 8

The full walk goes from (g f a e)
Direct edge goes from (g e)

Due to triangle inequality:
$c(g,e) \leq c(g,f) + c(f,a) + c(a,e)$
$8 \leq 10$

Thus,

$$c(H) \leq c(W) \leq 2c(H^*)$$

IMPROVING THE TRAVELING SALESPERSON BOUND

Claim: There exists an algorithm that improves the approximation ratio to $\frac{3}{2}$

Claim: There exists an algorithm that improves the approximation ratio to $\frac{3}{2}$

Christofides-TSP(G,c):

1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree vertices of T, and set $E = T \cup M$, doubling edges as needed.
3. Perform a Eulerian Tour T' of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the order of their first appearance in T'
5. return H

It is not obvious if / why this works. We need to define some terms and ANALYZE!

```
Christofides-TSP(G,c):
1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree
   vertices of T, and set E = T ∪ M, doubling edges as needed.
3. Perform a Eulerian Tour T′ of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the
   order of their first appearance in T′
5. return H
```

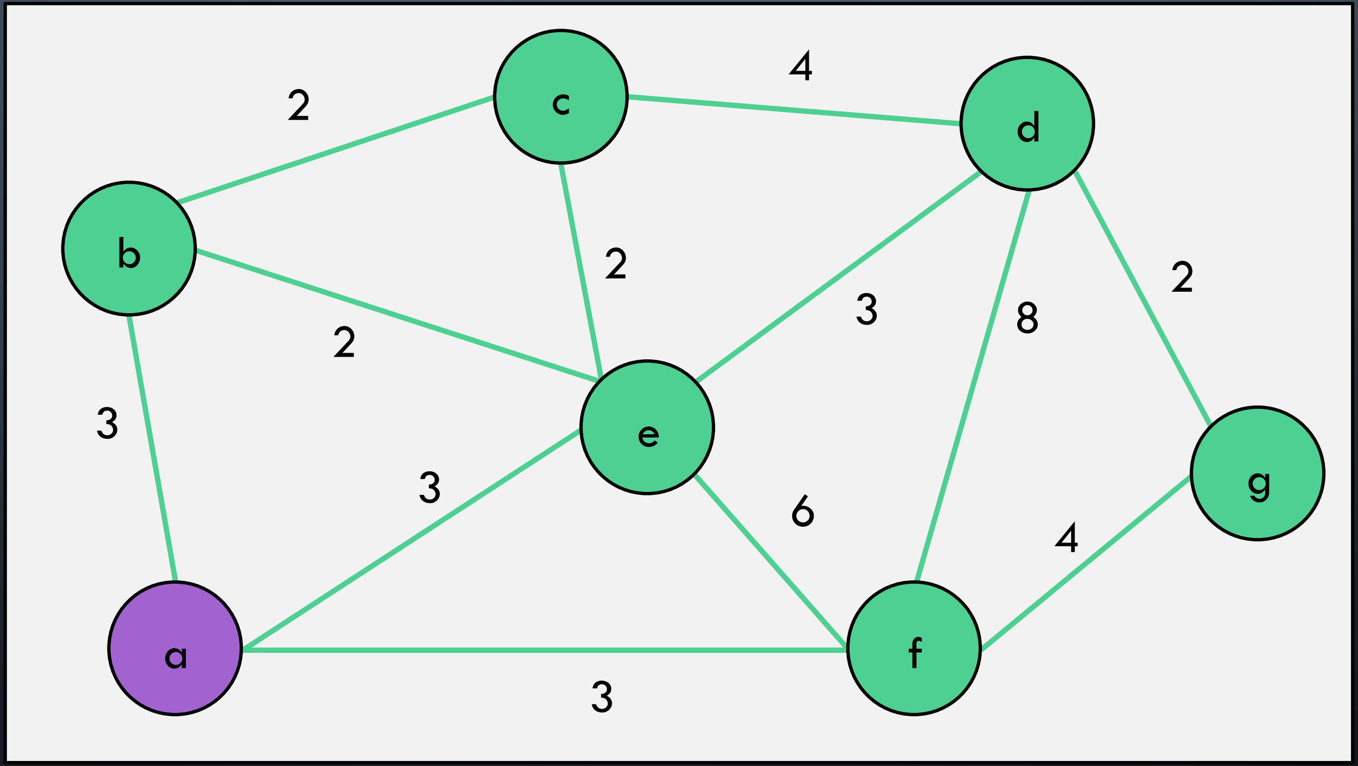
Eulerian Tour: Traversal of a graph that visits every edge of the graph exactly once, potentially visiting nodes multiple times, and returning back to the starting node.

Minimum Cost Perfect Matching: Given a set of nodes, choose edges such that each node is incident to exactly one chosen edge. Minimize total cost of chosen edges.

***Christofides-TSP(G,c)*:**

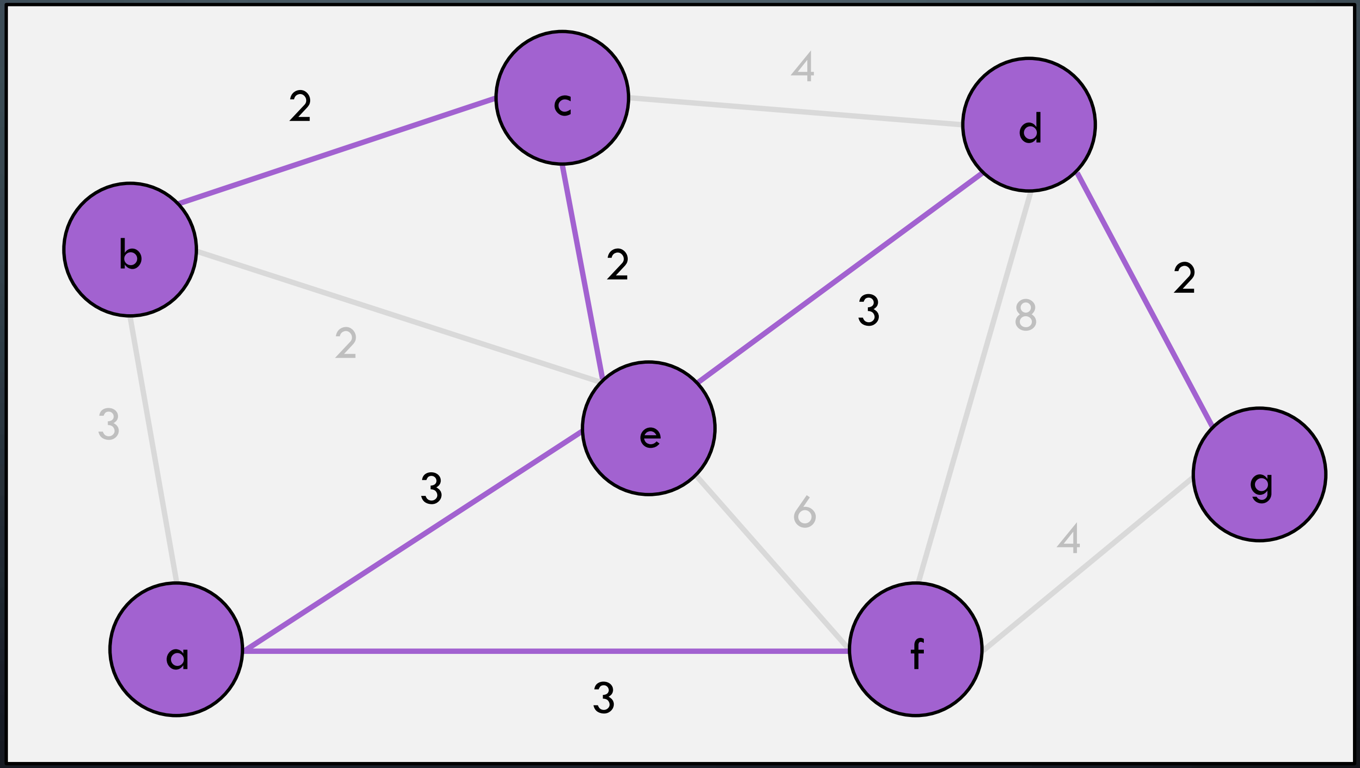
1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree vertices of T, and set $E = T \cup M$, doubling edges as needed.
3. Perform a Eulerian Tour T' of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the order of their first appearance in T'
5. return H

Assume missing edges have very high cost and graph is actually complete



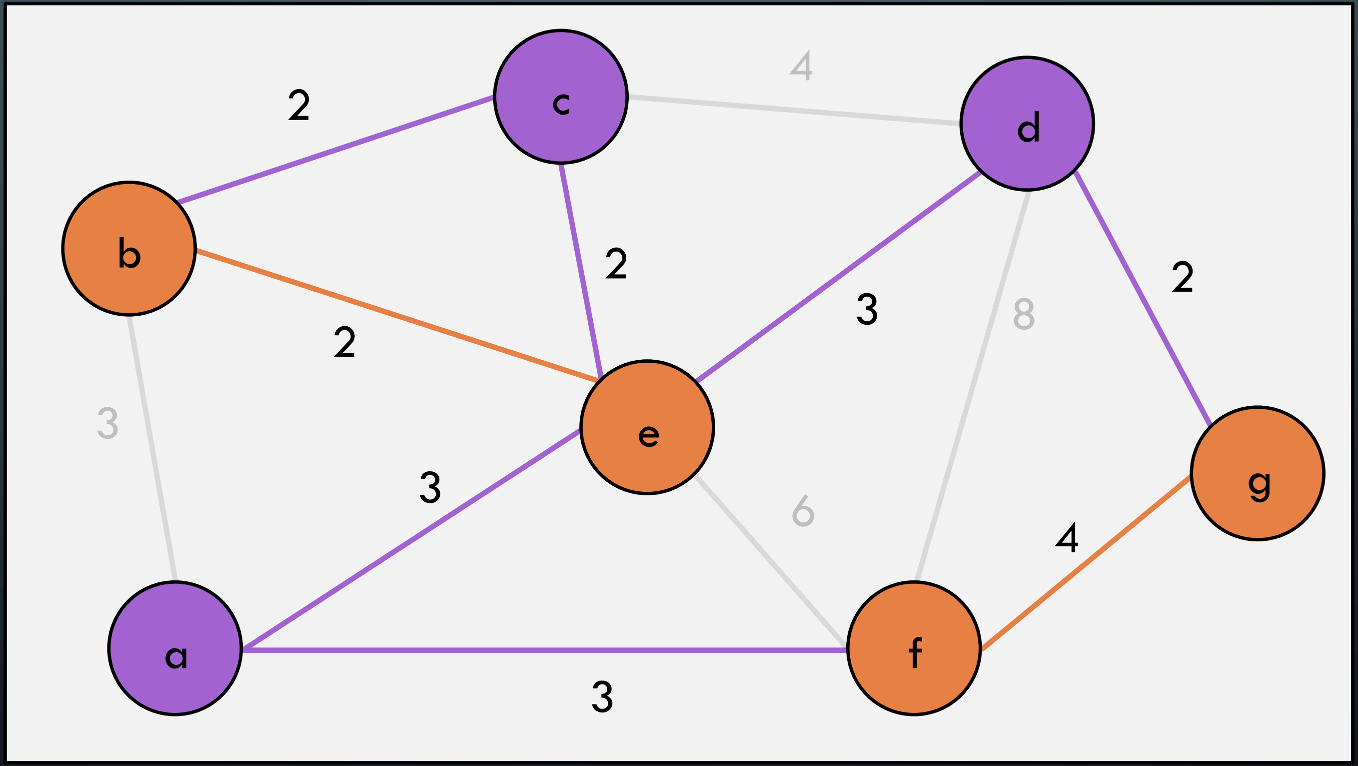
Christofides-TSP(G,c):

1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree vertices of T, and set $E = T \cup M$, doubling edges as needed.
3. Perform a Eulerian Tour T' of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the order of their first appearance in T'
5. return H



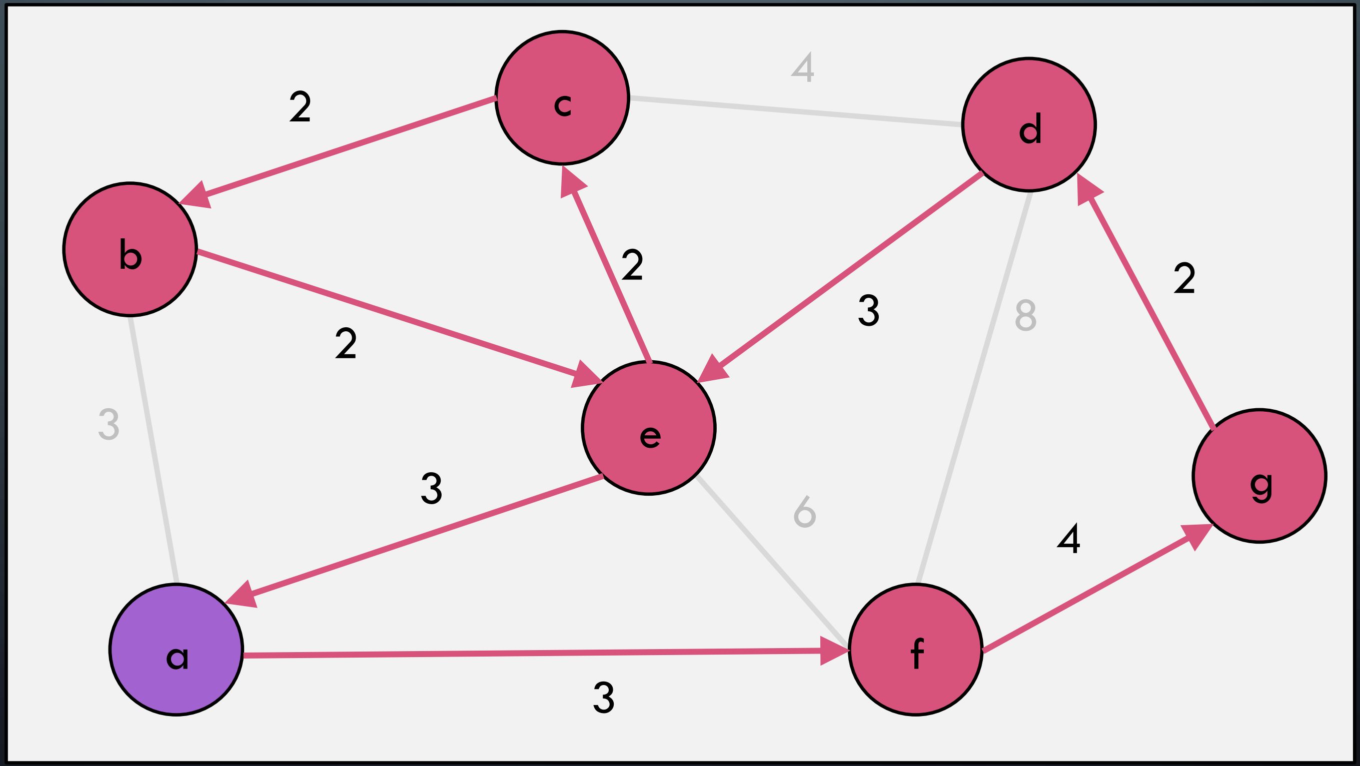
Christofides-TSP(G,c):

1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree vertices of T, and set $E = T \cup M$, doubling edges as needed.
3. Perform a Eulerian Tour T' of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the order of their first appearance in T'
5. return H



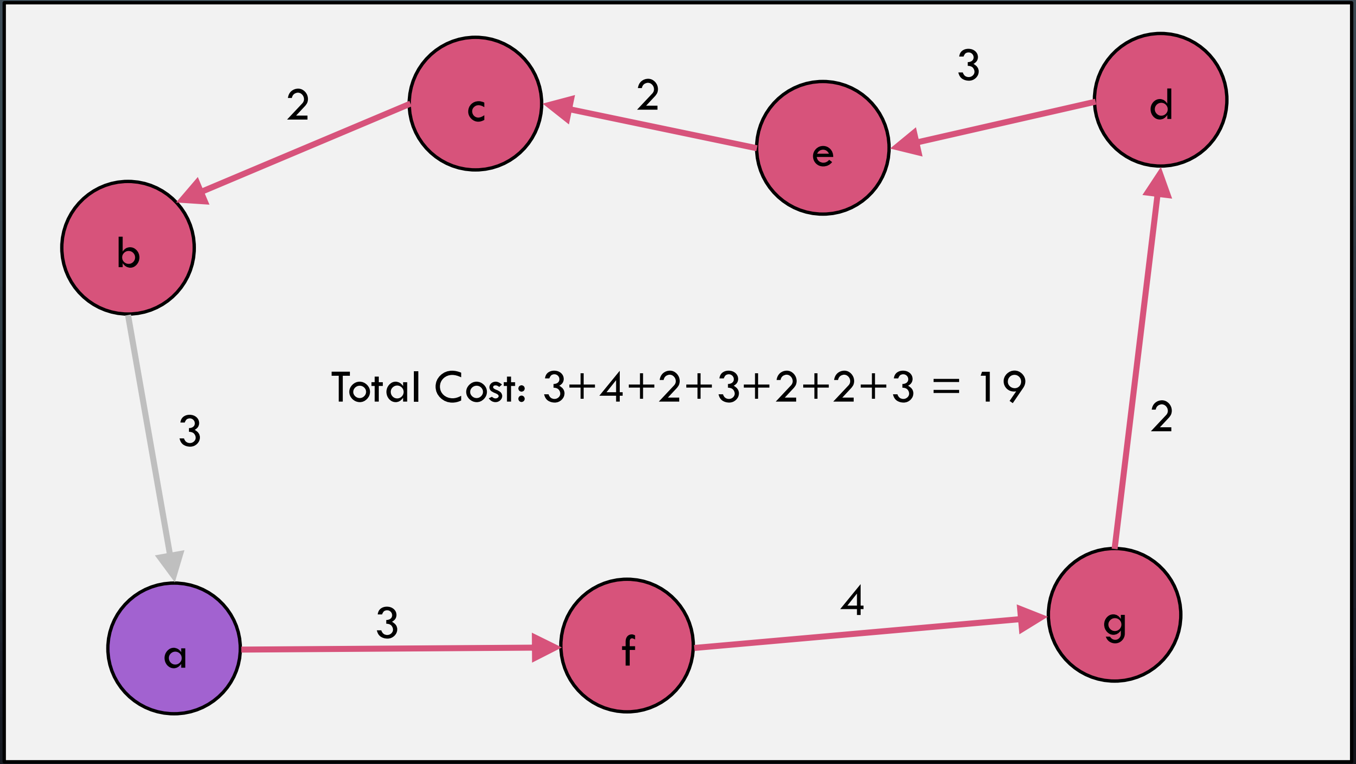
***Christofides-TSP(G,c)*:**

1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree vertices of T, and set $E = T \cup M$, doubling edges as needed.
3. Perform a Eulerian Tour T' of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the order of their first appearance in T'
5. return H



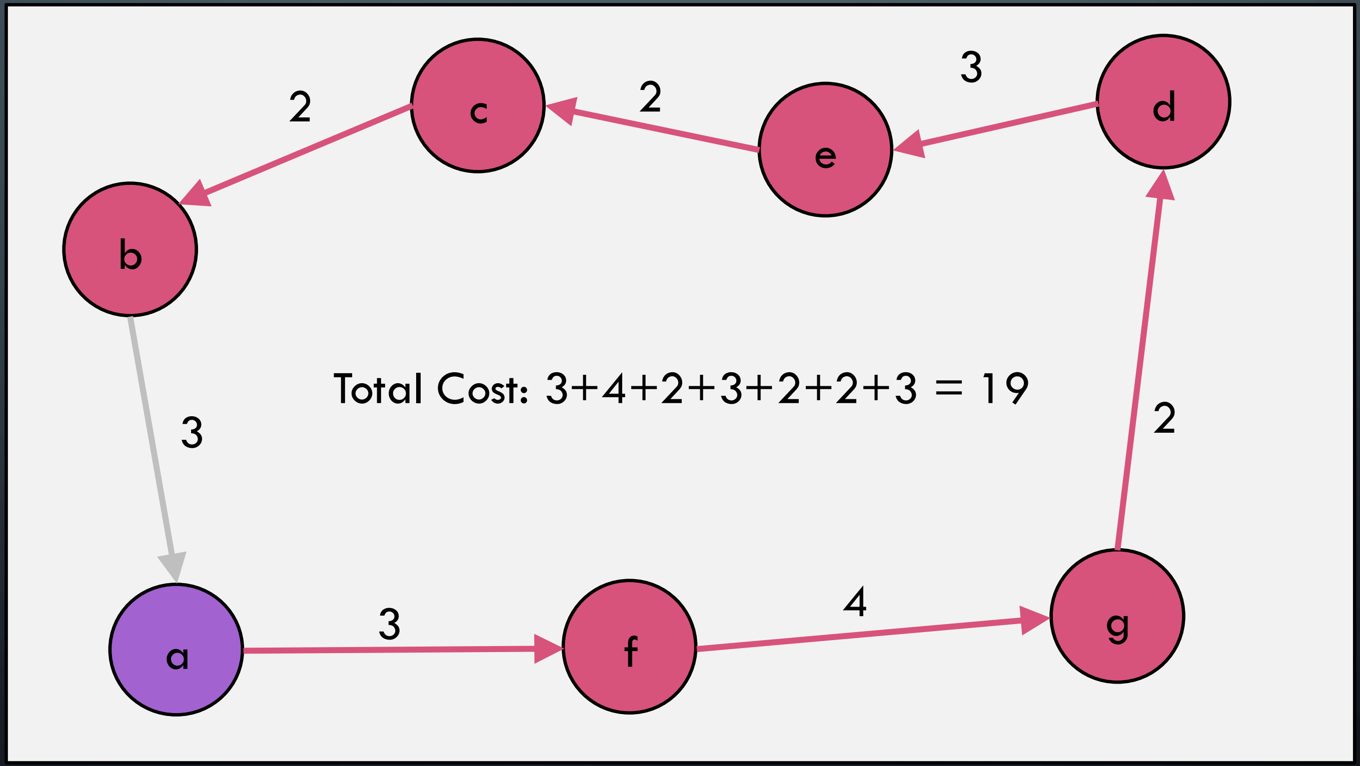
Christofides-TSP(G,c):

1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree vertices of T, and set $E = T \cup M$, doubling edges as needed.
3. Perform a Eulerian Tour T' of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the order of their first appearance in T'
5. return H



Christofides-TSP(G,c):

1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree vertices of T, and set $E = T \cup M$, doubling edges as needed.
3. Perform a Eulerian Tour T' of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the order of their first appearance in T'
5. return H



This happens to be the optimal solution for this particular graph

Claim: Christofides-TSP has an approximation ratio to $\frac{3}{2}$

Lemma 1: The number of odd degree vertices in any undirected graph must be even.

Do you see why? I will argue this verbally

Claim: Christofides-TSP has an approximation ratio to $\frac{3}{2}$

Christofides-TSP(G,c):

1. Compute a minimum spanning tree T of G
2. Compute a minimum cost perfect matching M on the odd-degree vertices of T, and set $E = T \cup M$, doubling edges as needed.
3. Perform a Eulerian Tour T' of E
4. Construct a Hamiltonian Cycle H by listing the vertices of G in the order of their first appearance in T'
5. return H

Lemma 1: The number of odd degree vertices in any undirected graph must be even.

So this means:

1. The MST T is guaranteed to have an even number of odd-degree nodes.
2. Since the graph is complete, the least-cost perfect matching M of these nodes must exist.

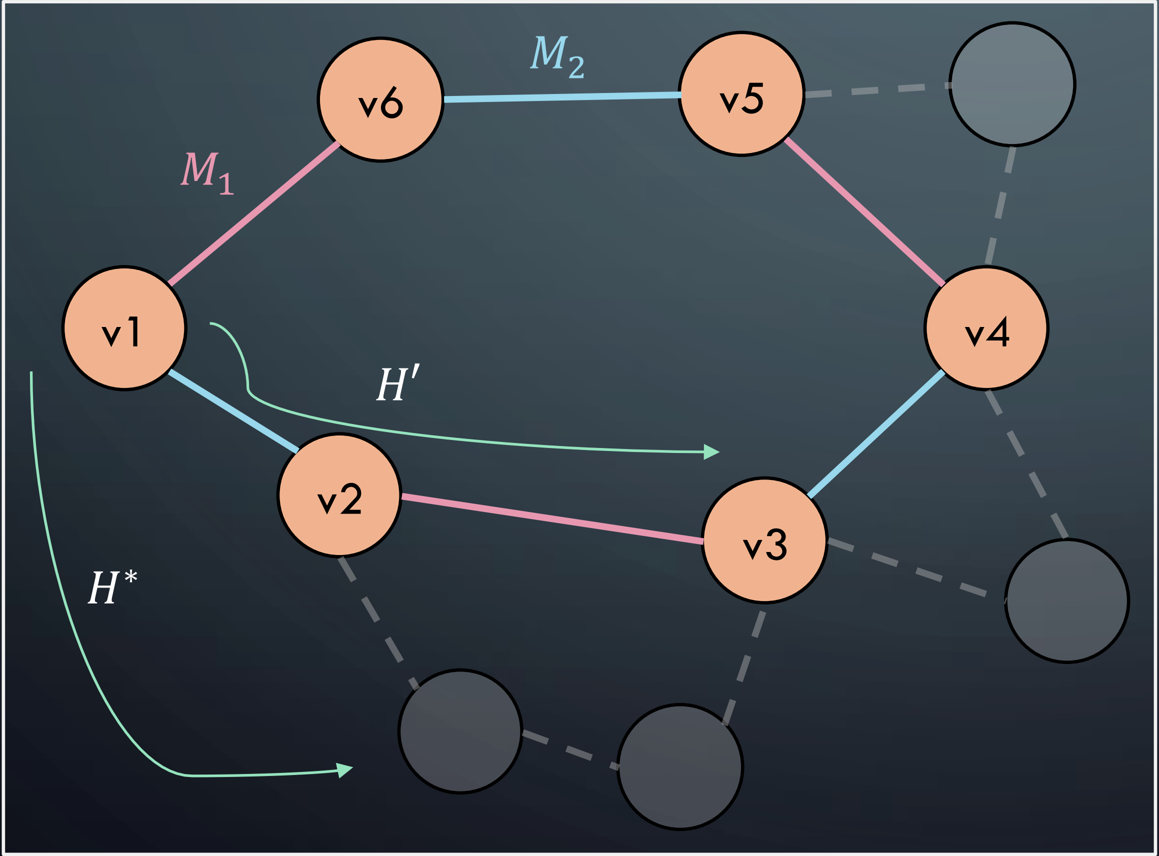
Claim: Christofides-TSP has an approximation ratio to $\frac{3}{2}$

Lemma 2: For any perfect min-cost matching M on an even number of nodes, we have $c(M) \leq \frac{c(H^*)}{2}$

Claim: Christofides-TSP has an approximation ratio to $\frac{3}{2}$

Lemma 2: For min-cost perfect matching M, we have

$$c(M) \leq \frac{c(H^*)}{2}$$



1. Take optimal tour H^* and create tour H' by taking the nodes in nodes with odd-degree in the order they appear in H^*, note that $c(H') \leq c(H^*)$
2. Split H' into two matchings M_1 and M_2 with alternating edges.
3. Note that M_1 and M_2 are both matchings, but may not be the least cost one.
4. $c(M) \leq \min\big(c(M_1), c(M_2)\big) \leq \frac{c(H')}{2} \leq \frac{c(H^*)}{2}$

Claim: Christofides-TSP has an approximation ratio to $\frac{3}{2}$

Lemma 1: The number of odd degree vertices in any undirected graph must be even.

Lemma 2: For m.c. perf. M, we have $c(M) \leq \frac{c(H^*)}{2}$

$$\underbrace{c(H) \leq c(T')}_{\text{By triangle inequality}} \underbrace{= c(T) + c(M)}_{\text{By how T' is constructed}} \underbrace{\leq c(H^*) + \frac{c(H^*)}{2}}_{\text{MST less than optimal tour + by lemma 2}} \leq \frac{3}{2} c(H^*)$$

APPROXIMATING THE GENERAL TRAVELING-SALESPERSON PROBLEM

APPROXIMATING THE GENERAL TSP

Question: Our TSP approximations assume connected graphs and triangle inequality. Can we approximate the general TSP without these assumptions?

What do you think?

APPROXIMATING THE GENERAL TSP

<u>Claim</u>: The general TSP is $\rho(n)$ approximatable in polynomial time if and only if $P = NP$

Claim: The general TSP is $\rho(n)$ approximatable in polynomial time if and only if $P = NP$

Proof Overview:

Assume there exists a $\rho(n)$ approximation for the general TSP problem

→

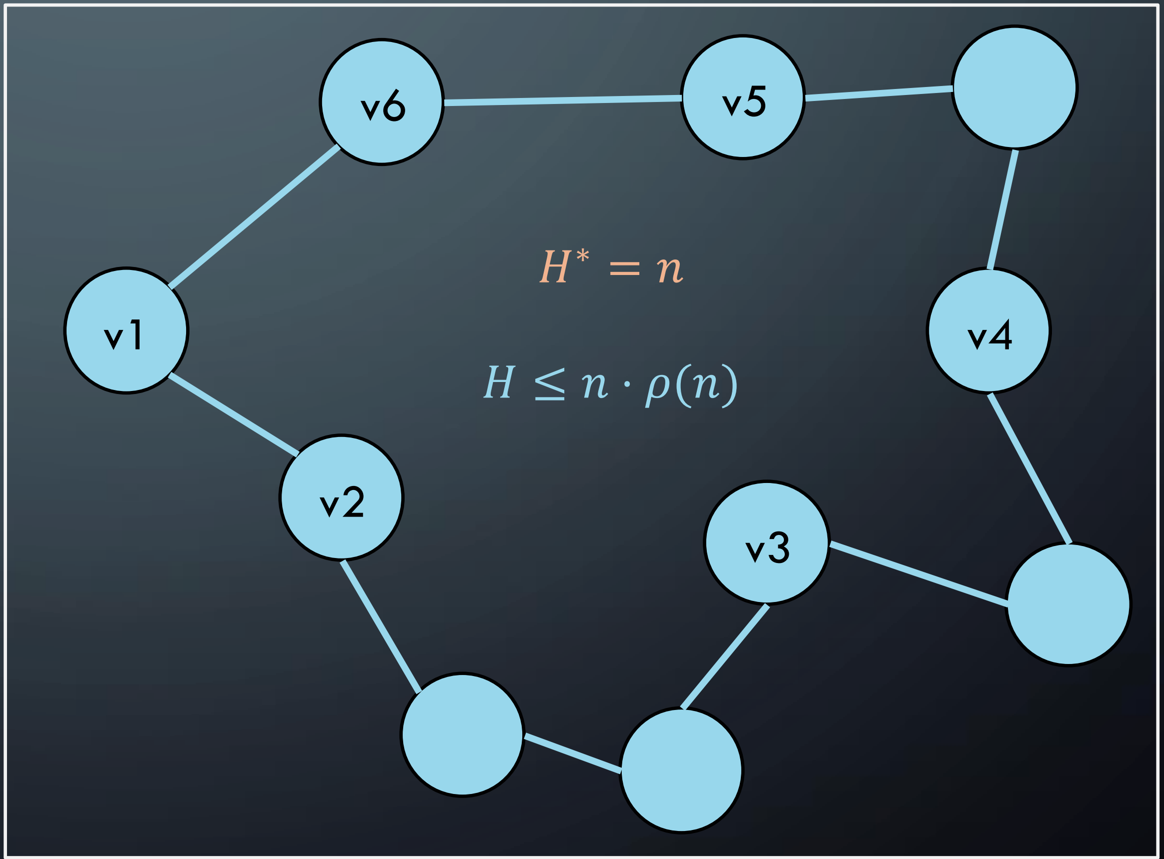
Choose / find a known NP-Complete Problem K

→

Show that $K \leq_p ATSP$: that an approximation to TSP (ATSP) can be used to solve K

Claim: The general TSP is $\rho(n)$ approximatable in polynomial time if and only if $P = NP$

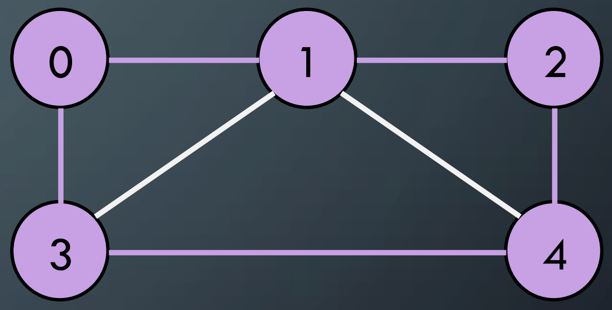
Assume there exists a $\rho(n)$ approximation for the general TSP problem



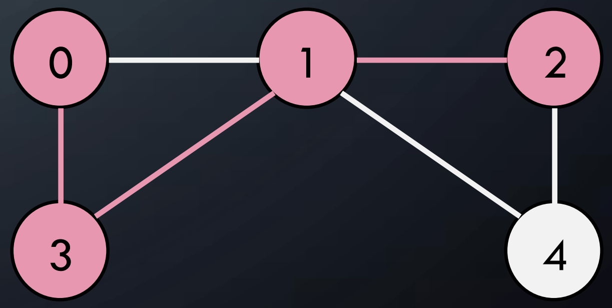
Claim: The general TSP is $\rho(n)$ approximatable in polynomial time if and only if $P = NP$

Hamiltonian Cycle: Given a graph G, does there exist a path that starts at some vertex, visits every node exactly once, and returns to the original starting vertex?

Choose / find a known NP-Complete Problem K



Cycle: Yes!



Cycle: No (one attempt shown)

Claim: The general TSP is $\rho(n)$ approximatable in polynomial time if and only if $P = NP$

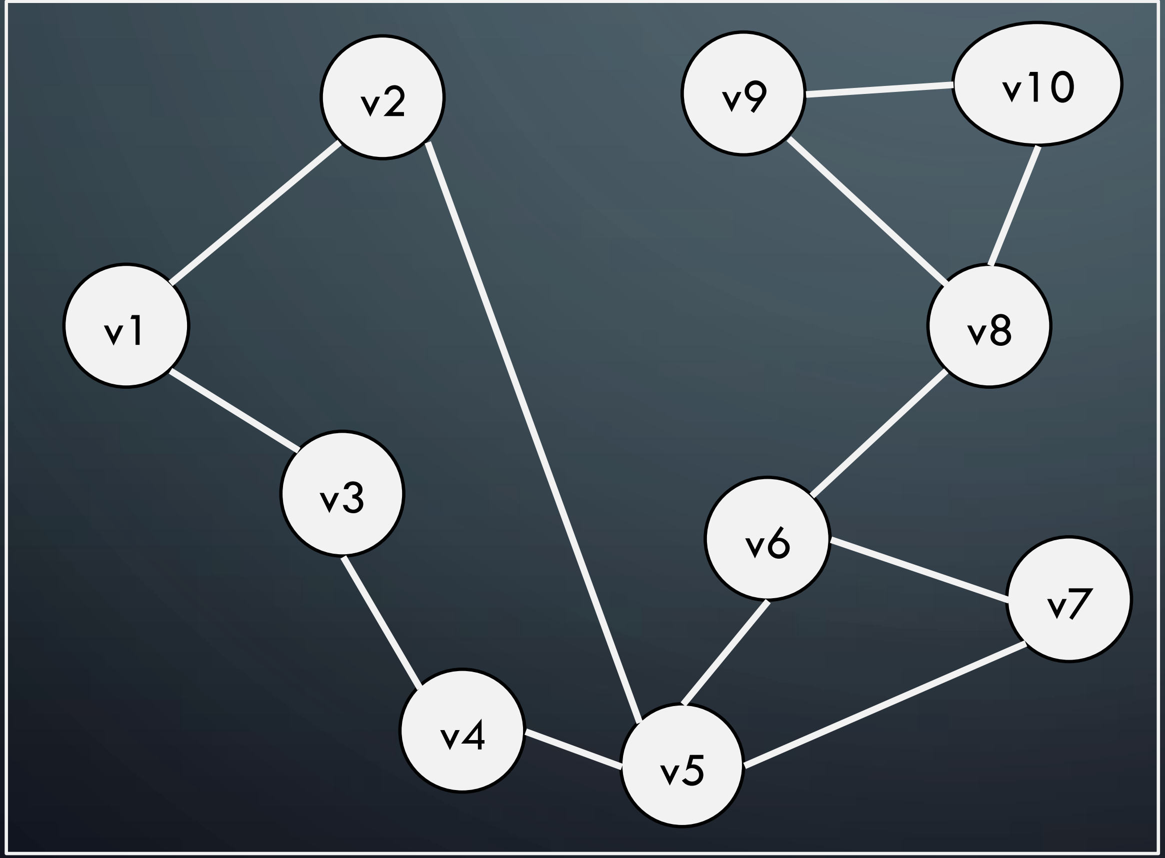
Approach:

Given an instance of Hamiltonian Cycle (HC), construct a graph that has a Hamiltonian Cycle if and only if that graph is approximatable within a factor of $\rho(n)$

Show that $\mathrm{HC} \leq_p ATSP$: that an approximation to TSP (ATSP) can be used to solve K

Claim: The general TSP is $\rho(n)$ approximatablein polynomial time if and only if $P = NP$

Show that HC $\leq_p ATSP$:



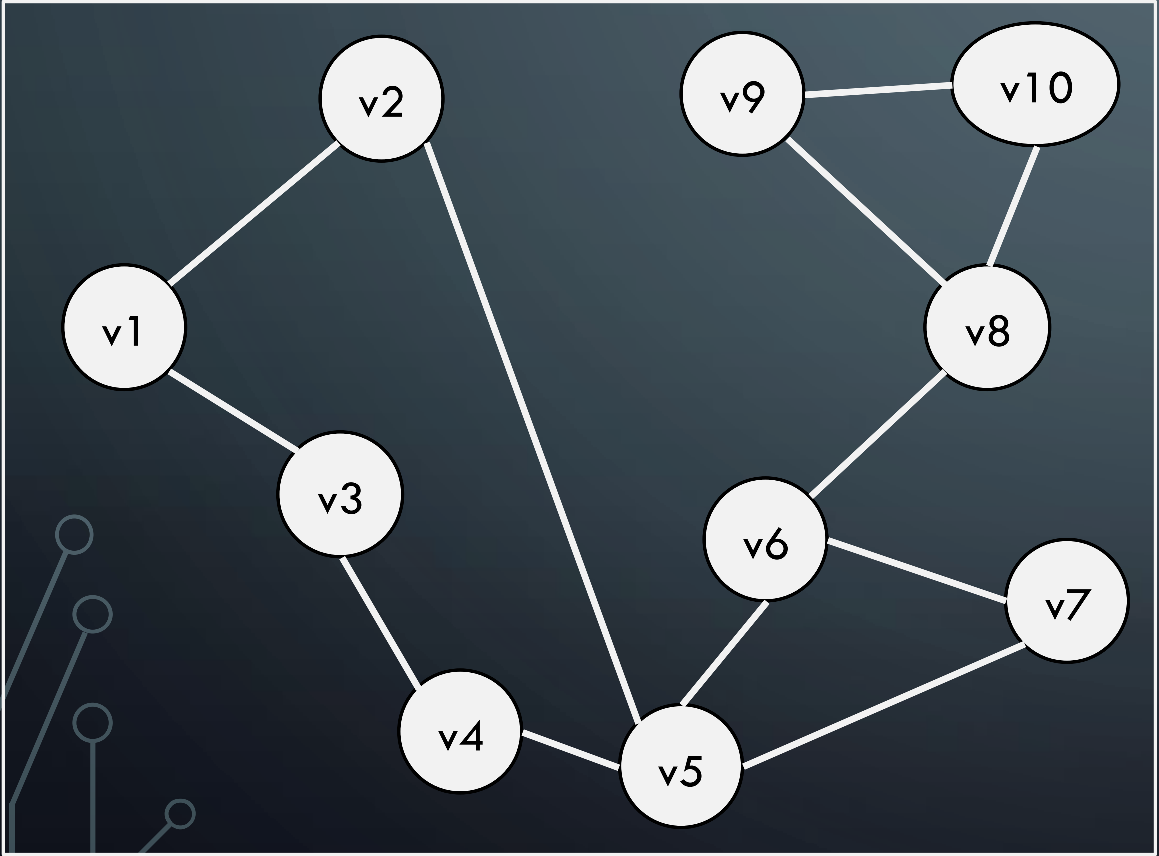
We are given a graph G, for which we want to find a Hamiltonian cycle. Note the answer to this one is NO

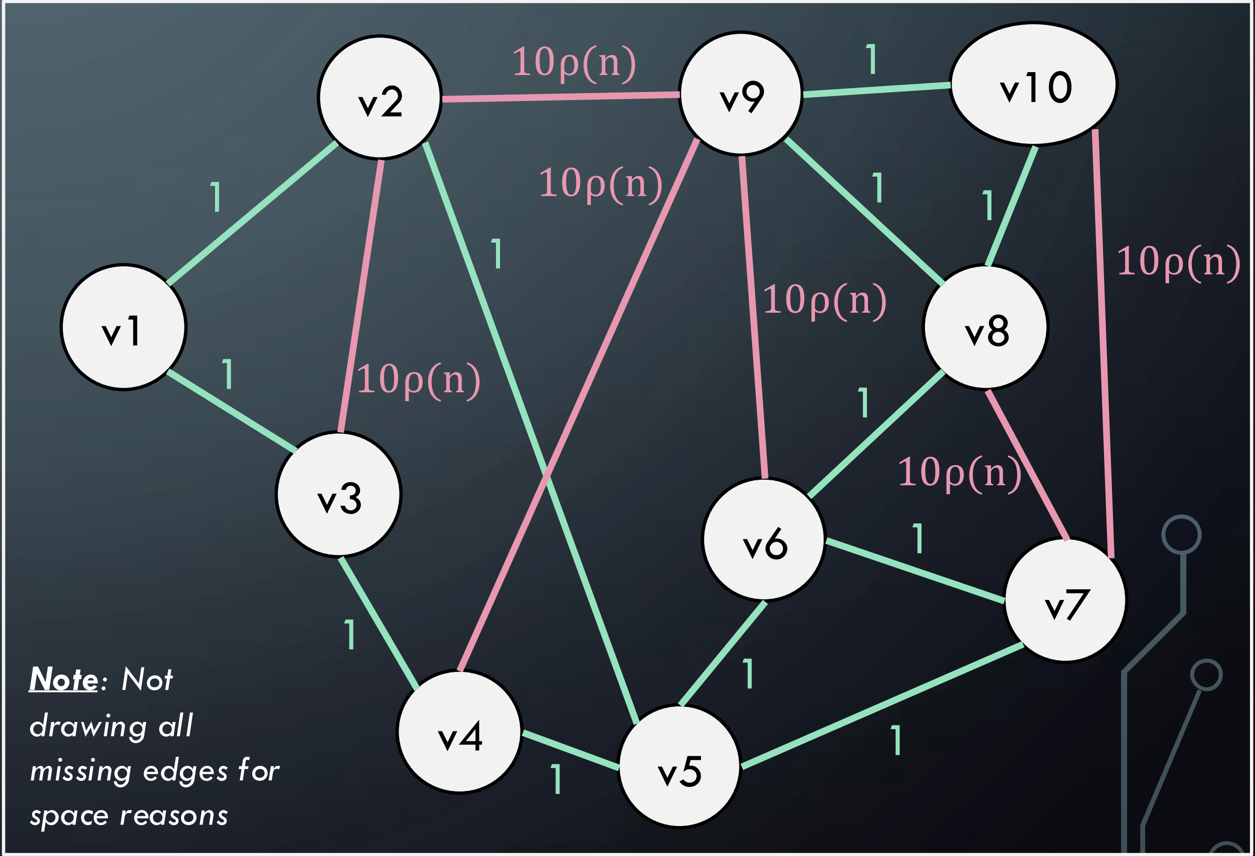
Our target TSP tour cost is going to be n, this means each edge incurred exactly cost 1

Claim: The general TSP is $\rho(n)$ approximatable in polynomial time if and only if $P = NP$

Convert the graph into an instance of TSP as shown here. For each edge, add cost of 1, for each missing edge, add cost of $n \cdot \rho(n)$ where $n = |V|$

$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ n \cdot \rho(n) & \text{if } (u,v) \notin E \end{cases}$$

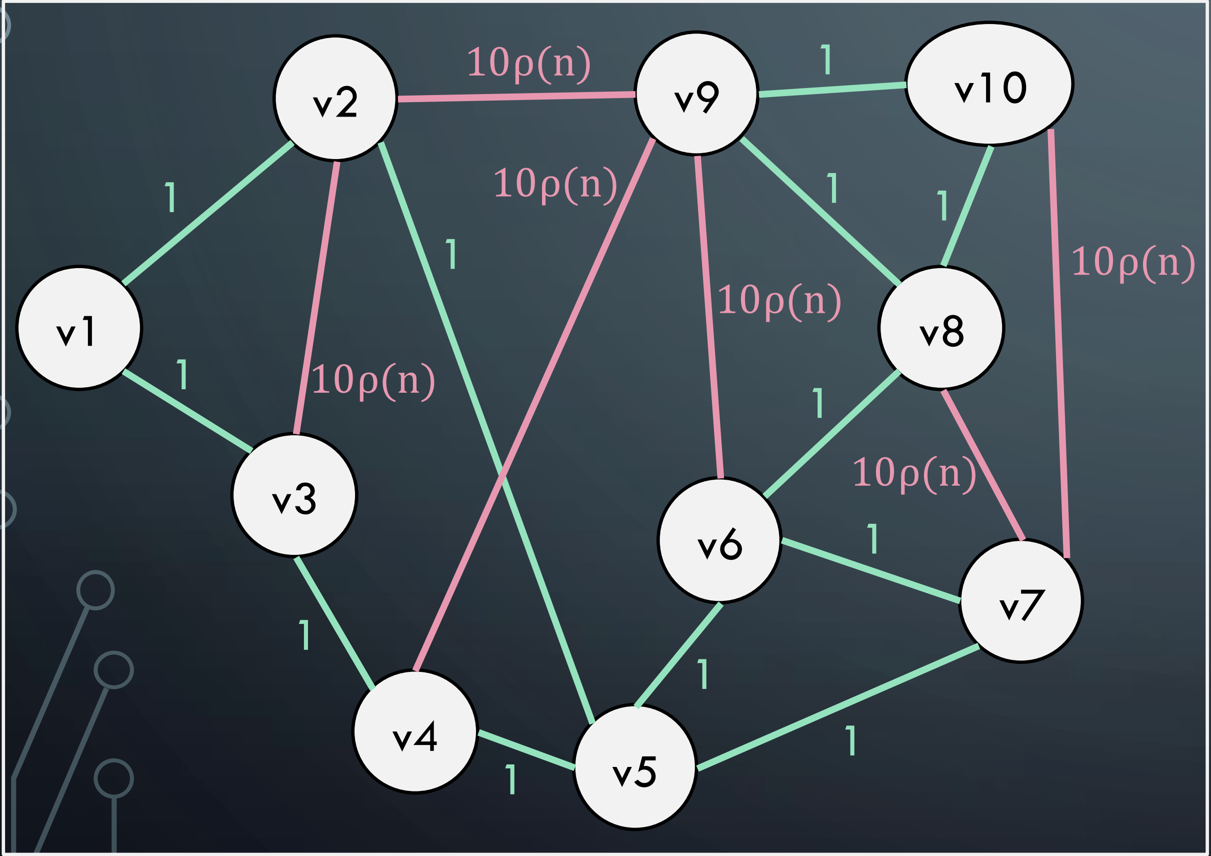




Note: *Not drawing all missing edges for space reasons*

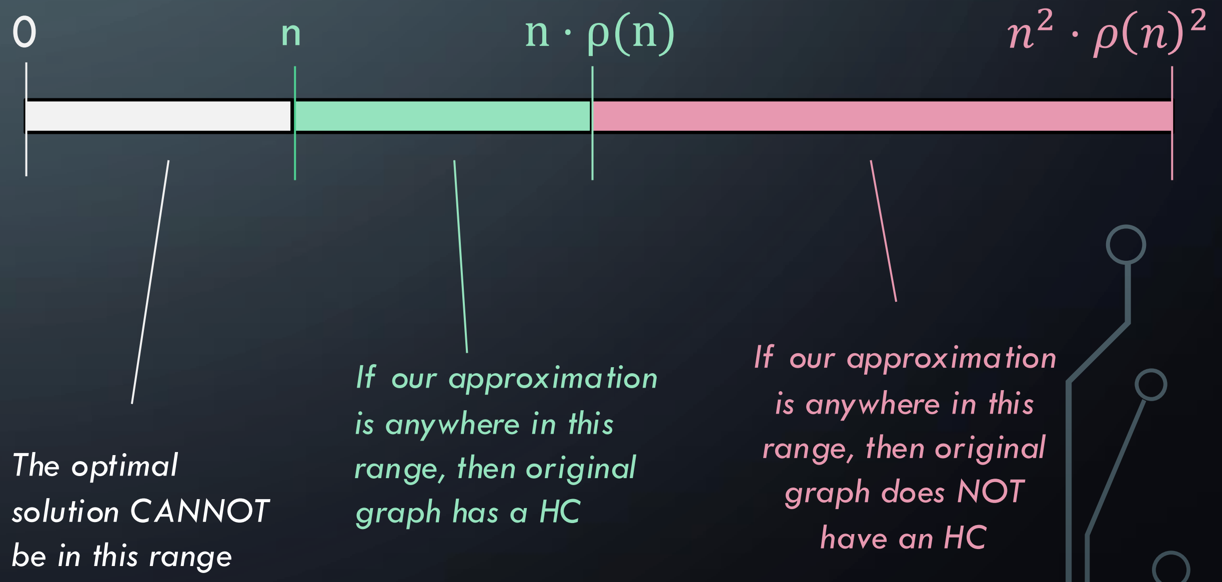
Claim: The general TSP is $\rho(n)$ approximatable in polynomial time if and only if $P = NP$

$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ n \cdot \rho(n) & \text{if } (u,v) \notin E \end{cases}$$



This new graph has two possible optimal TSP tours:

n	if original graph has a HC
$> n \cdot \rho(n)$	if original graph has no HC

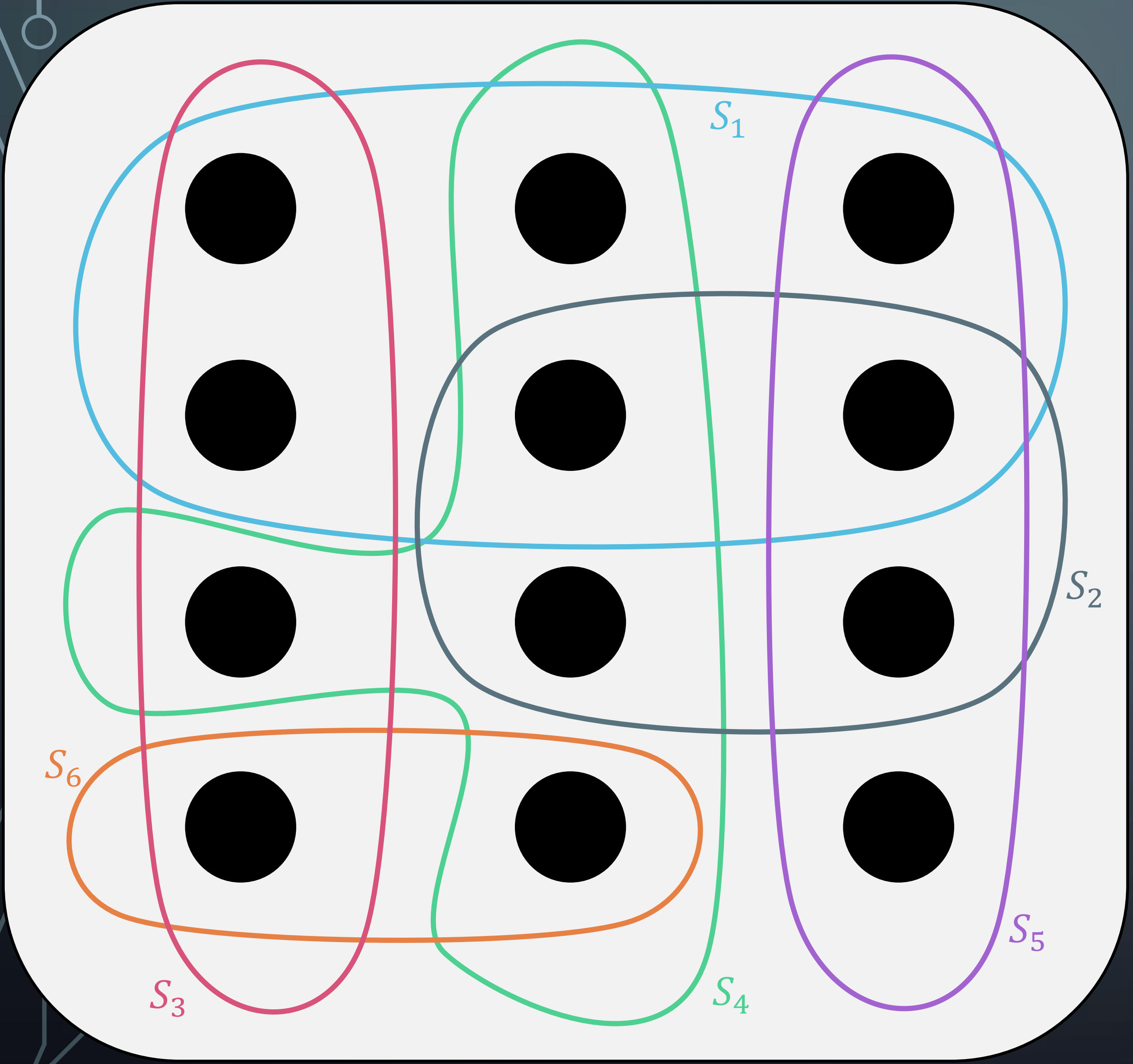


APPROXIMATING THE GENERAL TSP

Proven: The general TSP is $\rho(n)$ approximatable in polynomial time if and only if $P = NP$

SET COVER

SET COVER PROBLEM



Set Cover Problem Statement

INPUT:

A finite set X (black nodes in image)

A family F of subsets of X | $X = \bigcup_{S \in F} S$

A subset S is said to ***cover*** its elements

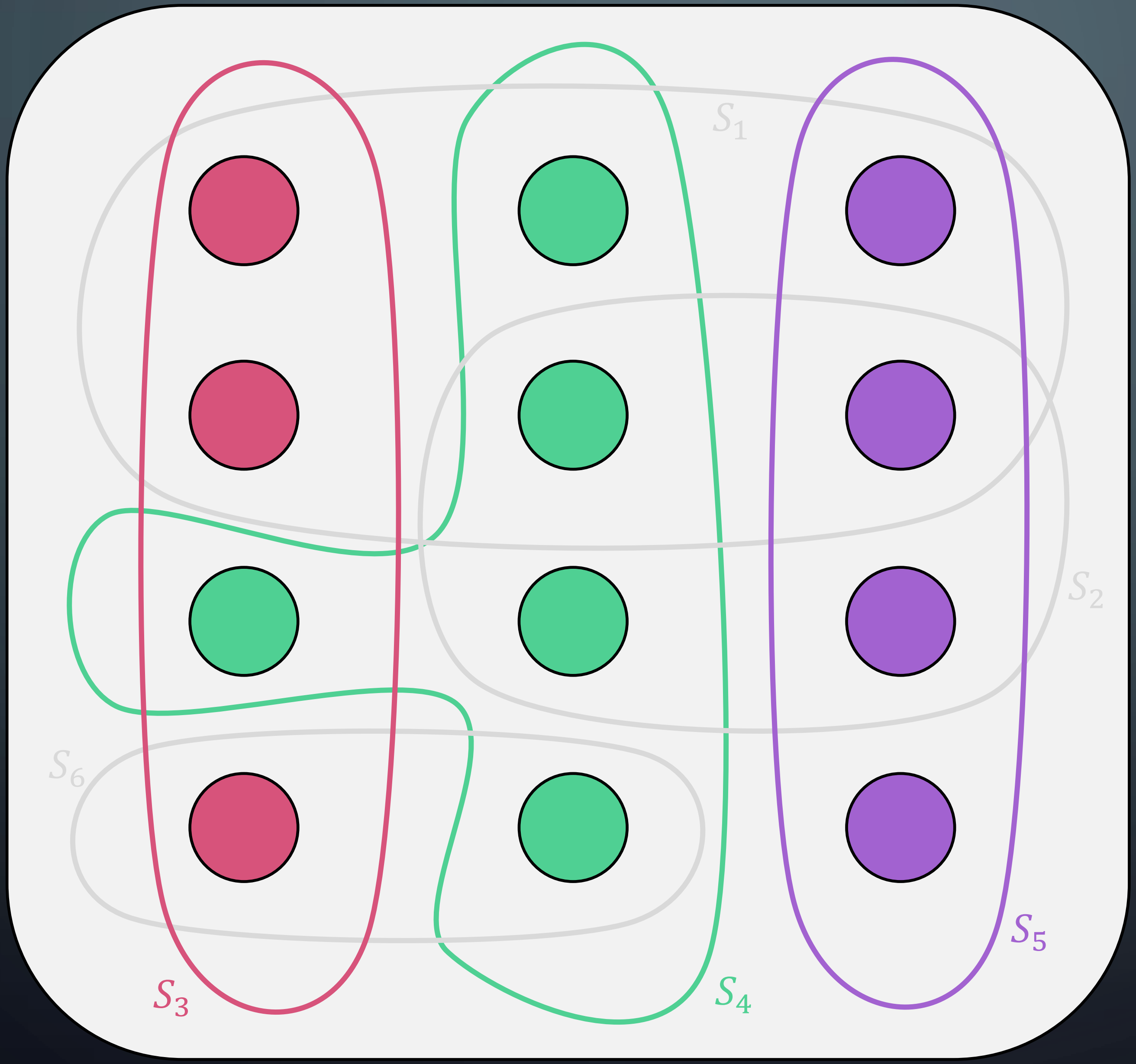
OUTPUT:

Minimum size subset $C \subseteq F$ such that:

$X = \bigcup_{S \in C} S$

In other words, find the smallest group of subsets that include all the elements.

SET COVER PROBLEM



Minimum cover is called C^*, for this case, C^* is shown and $|C^*| = 3$

Set Cover is also NP-Hard, by the way.

APPROXIMATING SET COVER

A greedy approximation for set cover

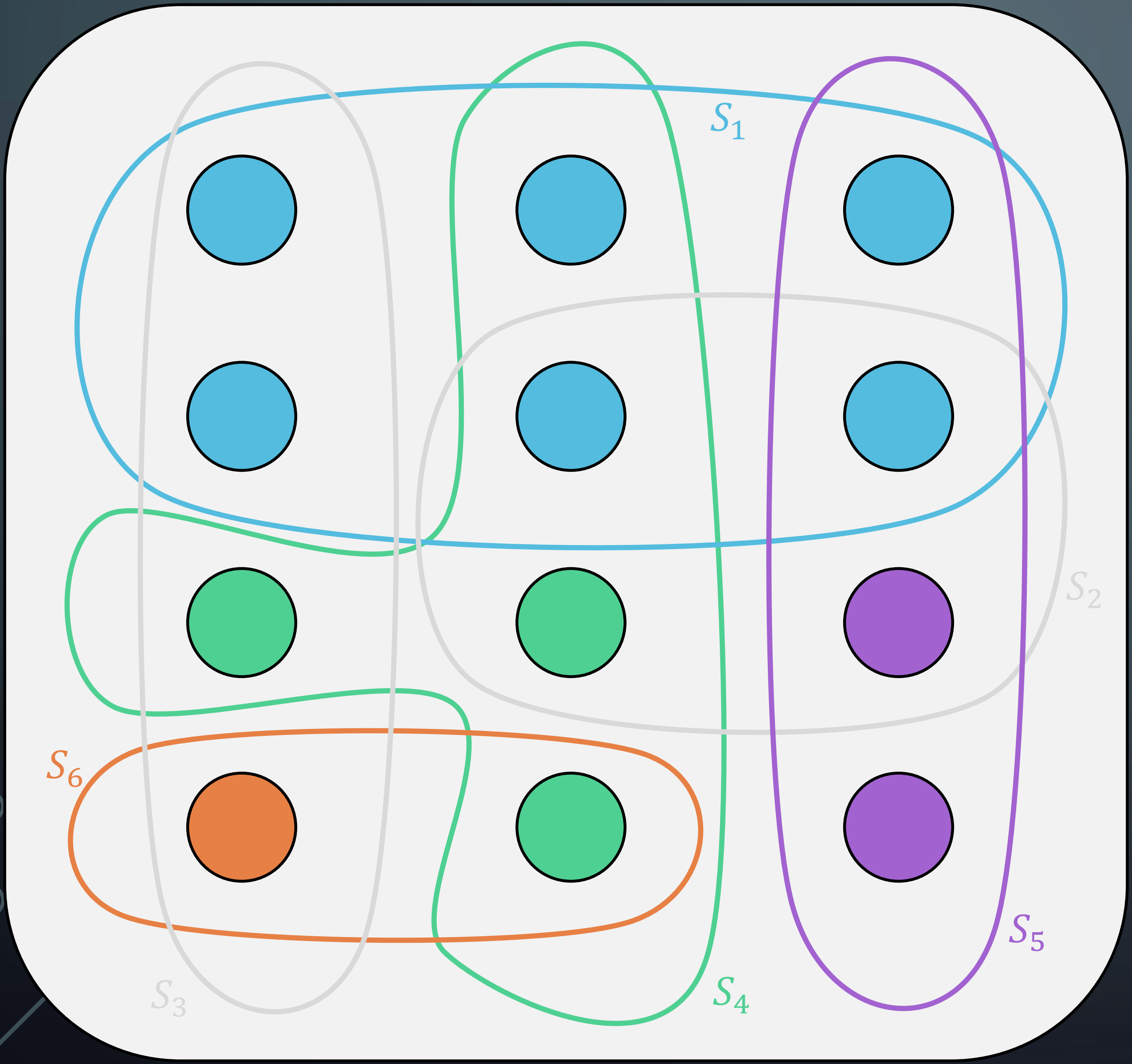
Greedy-Set-Cover(X, F):
    $U = X$
    $C = \emptyset$
    while $U \neq \emptyset$:
        select an $S \in F$ that maximizes $|S \cap U|$
        $U = U - S$
        $C = C \cup \{S\}$
    return C

How good of a cover will this produce? Any "gut" reactions?

… in other words, always select the subset that adds the most new (currently uncovered) elements to the cover

What is the runtime of this method?

APPROXIMATING SET COVER



A greedy approximation for set cover

Greedy-Set-Cover(X, F):
    $U = X$
    $C = \emptyset$
    while $U \neq \emptyset$:
        select an $S \in F$ that maximizes $|S \cap U|$
        $U = U - S$
        $C = C \cup \{S\}$
    return C

Execution on test case gives:

Add S_1 first (adds six nodes to cover)
Add S_4 next (adds three nodes to cover)
Add S_5 next (adds two nodes to cover)
Add S_6 last (adds one node to cover)

$$|\boldsymbol{C}| = \mathbf{4} > |\boldsymbol{C}^*| = \mathbf{3}$$

ANALYSIS

Theorem:

Greedy-Set-Cover(X, F) is a $\rho(n)$-approximation of set-cover where

$$\rho(n) = H(\max(|S| : S \in F))$$

H() is a harmonic number
(see next slide)

ASIDE: HARMONIC NUMBERS

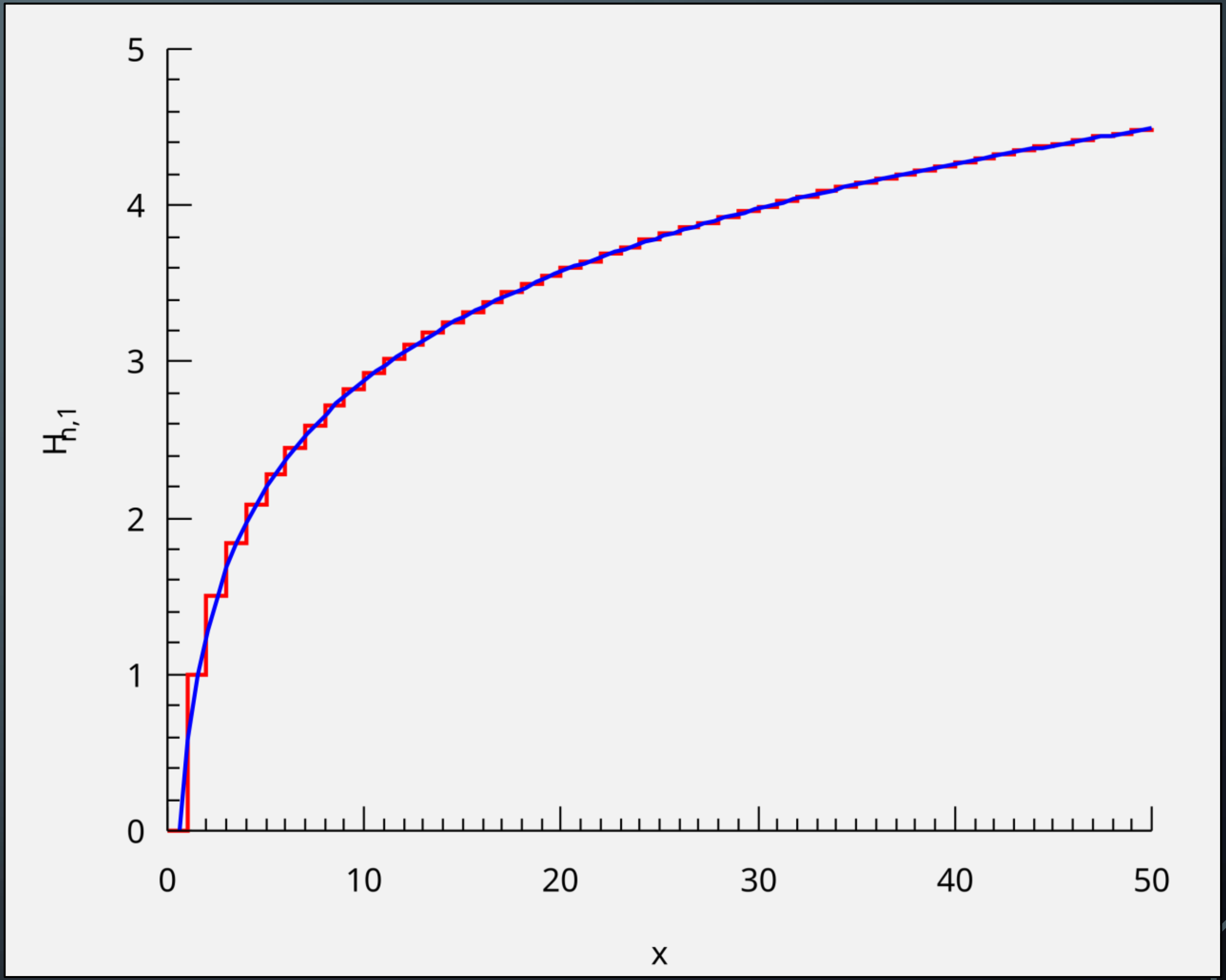
Harmonic Number:

$$H(d) = H_d = \sum_{i=1}^{d} \frac{1}{d} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \dots + \frac{1}{d}$$

Why? Well set cover algorithm seems to have a similar pattern. You keep adding a percentage of nodes that are left on each iteration.

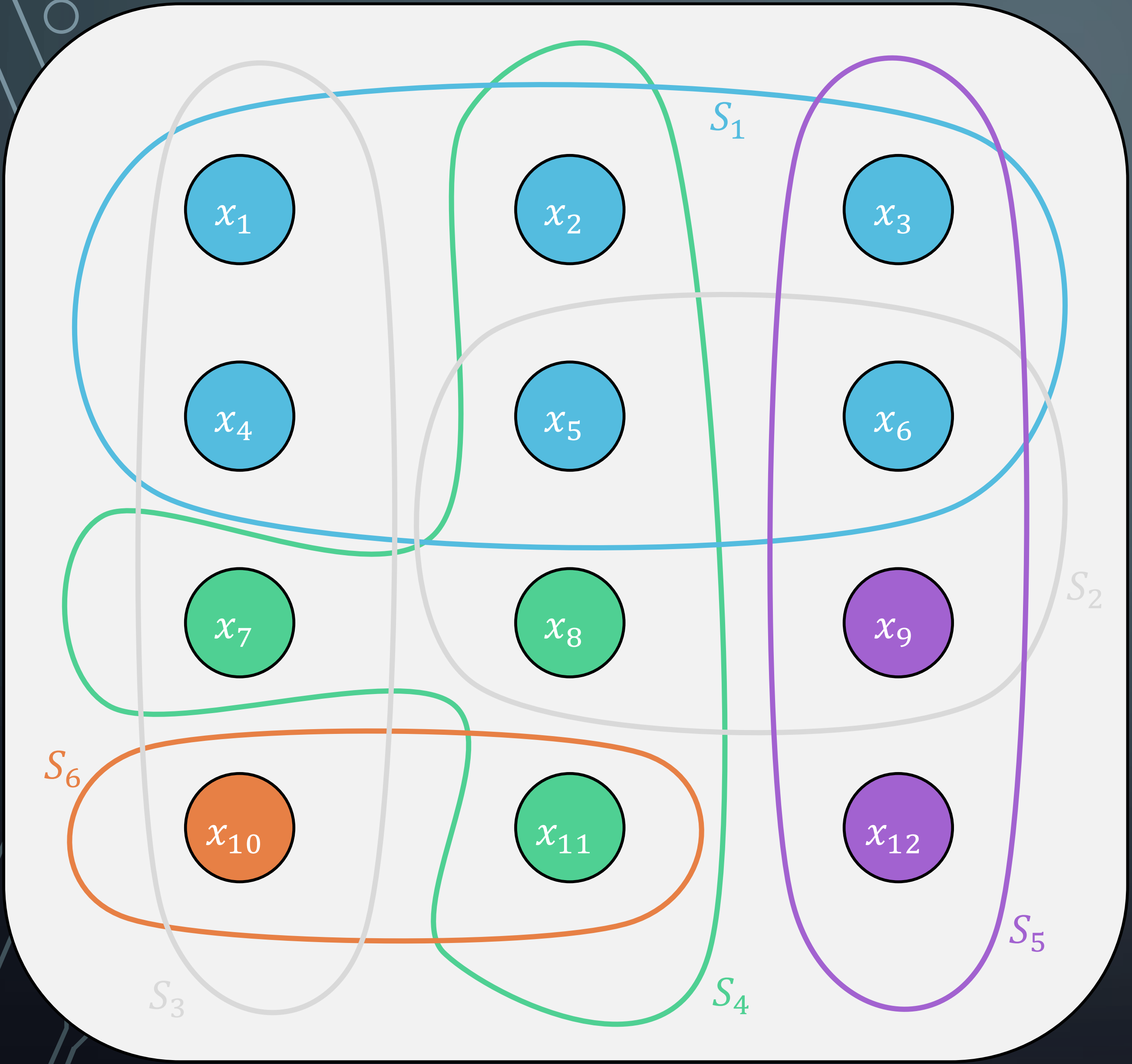
Note that it is established that:

$$H_d = \ln(d) + O(1)$$



*Harmonic number (x-axis) and value of summation H_d (y-axis). **source = Wikipedia*

ANALYSIS OF SET COVER GREEDY



Consider the following cost function

$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup S_3 \ \ldots \cup S_{i-1})|}$$

Each set we add to the solution adds a cost of 1, but spread cost out among elements added. Each element has exactly ONE cost.

Costs of the nodes in this example are:

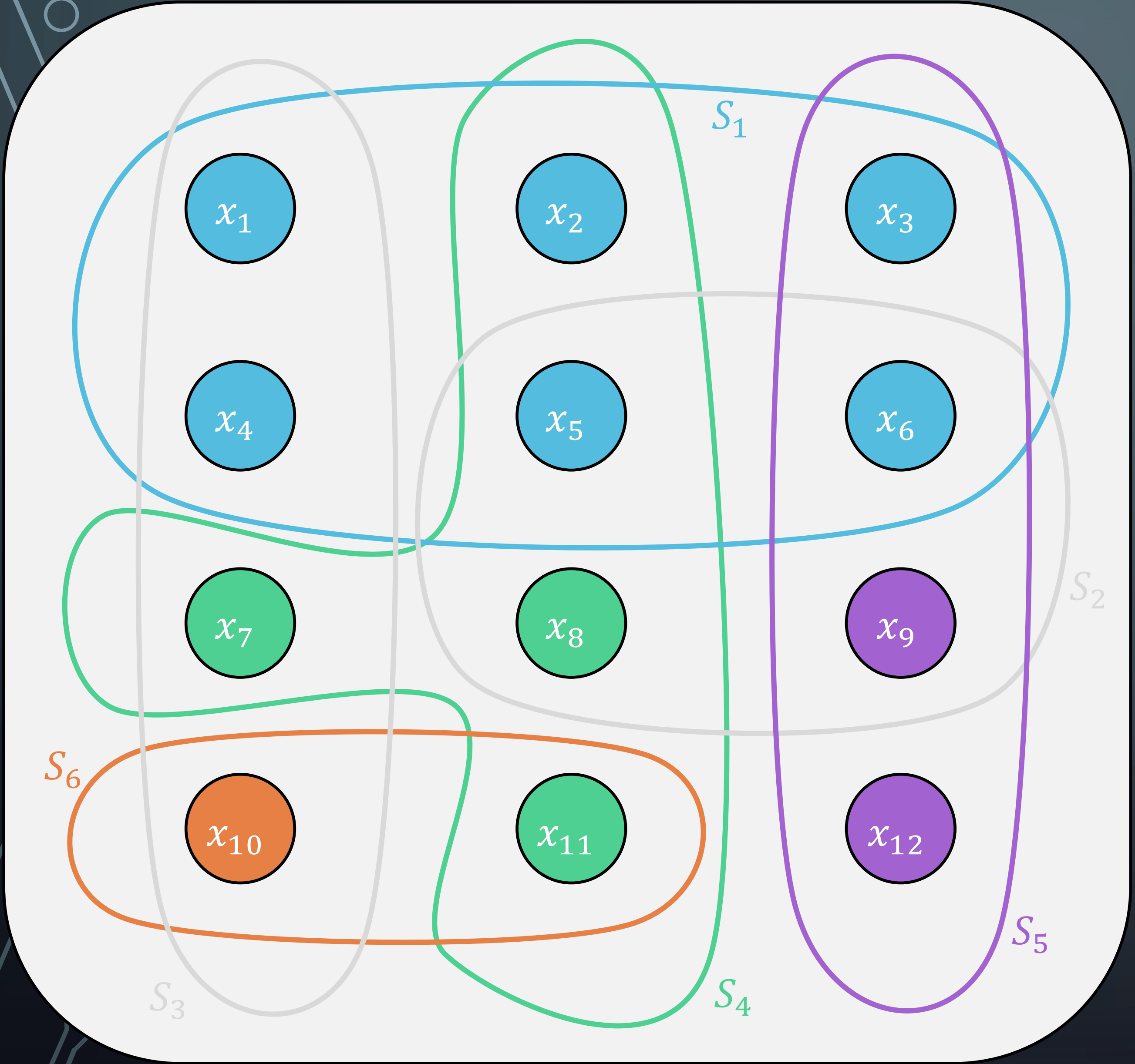
$$c_{x1} = c_{x2} = \cdots = c_{x6} = \frac{1}{6}$$

$$c_{x7} = c_{x8} = c_{x11} = \frac{1}{3}$$

$$c_{x9} = c_{x12} = \frac{1}{2}$$

$$c_{x10} = 1$$

ANALYSIS OF SET COVER GREEDY



Costs of the nodes in this example are:

$$c_{x1} = c_{x2} = \cdots = c_{x6} = \frac{1}{6}$$

$$c_{x7} = c_{x8} = c_{x11} = \frac{1}{3}$$

$$c_{x9} = c_{x12} = \frac{1}{2}$$

$$c_{x10} = 1$$

Note the following

$$|C| = \sum_{x \in X} c_x$$

Sum of costs equals total cost. In this case 4

$$\sum_{S \in C^*} \sum_{x \in S} c_x \geq \sum_{x \in X} c_x = |C|$$

Because left side duplicates some costs that right side does not

ANALYSIS OF SET COVER GREEDY

From Previous Slide

$$|C| \le \sum_{S \in C^*} \sum_{x \in S} c_x$$

$$\sum_{x \in S} c_x \le H(|S|)$$

Accept as true for now, will argue why in a moment

$$|C| \le \sum_{S \in C^*} H(|S|)$$

……………………

$$|C| \le |C^*| \times H(\max(|S| : S \in F))$$

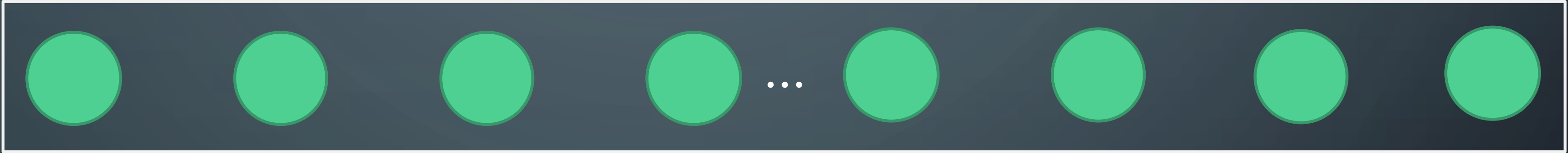
Q.E.D. if inequality assumed above is true

ANALYSIS OF SET COVER GREEDY

Analysis on previous slide relies on the following inequality. Does it actually hold?

$$\sum_{x \in S} c_x \leq H(|S|)$$

First, suppose that there is a set S with size |S| and algorithm adds all nodes from S "at once"



$$\sum_{x \in S} c_x = \quad \frac{1}{n} + \frac{1}{n} + \frac{1}{n} + \frac{1}{n} \cdots \frac{1}{n} + \frac{1}{n} + \frac{1}{n} + \frac{1}{n}$$

$$< \quad < \quad < \quad < \quad < \quad < \quad < \quad =$$

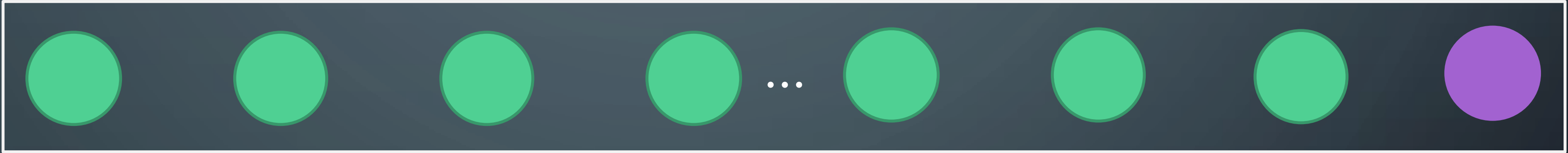
$$H(|S|) = \quad 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \cdots \frac{1}{n-3} + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$

ANALYSIS OF SET COVER GREEDY

Analysis on previous slide relies on the following inequality. Does it actually hold?

$$\sum_{x \in S} c_x \leq H(|S|)$$

Now, suppose one node gets added in the previous "round" and S gets added next.



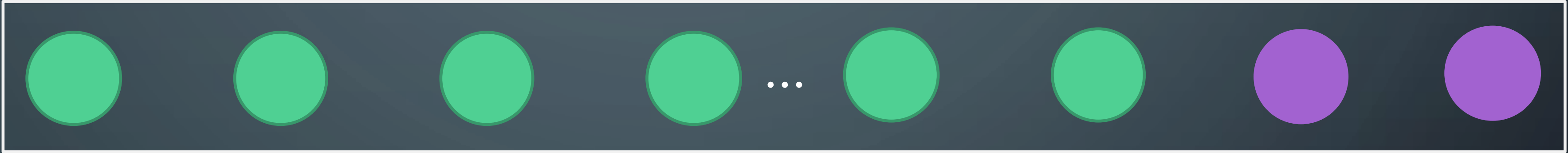
$$\begin{array}{cccccccccccccccc}
\sum_{x \in S} c_x = & \frac{1}{n-1} & + & \frac{1}{n-1} & + & \frac{1}{n-1} & + & \frac{1}{n-1} & \cdots & \frac{1}{n-1} & + & \frac{1}{n-1} & + & \frac{1}{n-1} & + & \frac{1}{n+\delta_1} \\
 & < & & < & & < & & < & & < & & < & & = & & < \\
H(|S|) = & 1 & + & \frac{1}{2} & + & \frac{1}{3} & + & \frac{1}{4} & \cdots & \frac{1}{n-3} & + & \frac{1}{n-2} & + & \frac{1}{n-1} & + & \frac{1}{n}
\end{array}$$

ANALYSIS OF SET COVER GREEDY

Analysis on previous slide relies on the following inequality. Does it actually hold?

Now, suppose S is added "third",
OR after two nodes already added.

$$\boxed{\sum_{x \in S} c_x \le H(|S|)}$$



$$\sum_{x \in S} c_x = \frac{1}{n-2} + \frac{1}{n-2} + \frac{1}{n-2} + \frac{1}{n-2} \cdots \frac{1}{n-2} + \frac{1}{n-2} + \frac{1}{n-1+\delta_2} + \frac{1}{n+\delta_1}$$

$<$ $<$ $<$ $<$ $<$ $=$ $<$ $<$

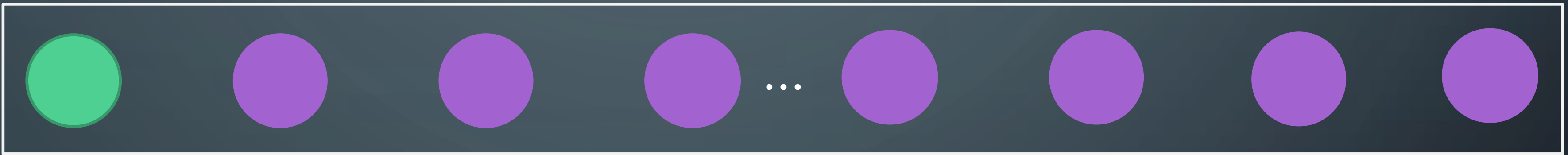
$$H(|S|) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \cdots \frac{1}{n-3} + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$

ANALYSIS OF SET COVER GREEDY

Analysis on previous slide relies on the following inequality. Does it actually hold?

Logic continues. What if last node is added by itself…

$$\sum_{x \in S} c_x \leq H(|S|)$$



$$\sum_{x \in S} c_x = \quad 1 + \frac{1}{2+\delta_7} + \frac{1}{3+\delta_6} + \frac{1}{4+\delta_5} \cdots \frac{1}{n-3+\delta_4} + \frac{1}{n-2+\delta_3} + \frac{1}{n-1+\delta_2} + \frac{1}{n+\delta_1}$$

$$= \quad < \quad < \quad < \quad < \quad < \quad < \quad <$$

$$H(|S|) = \quad 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \cdots \frac{1}{n-3} + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$

ANALYSIS

Theorem:

Greedy-Set-Cover(X, F) is a $\rho(n)$-approximation of set-cover where

$$\rho(n) = H(\max(|S| : S \in F)$$

Thus, it is proven!!

CONCLUSION

CONCLUSIONS

Basic Approximations:

- Are very fast (often greedy) algorithms. Algorithm / implementation not difficult.
- Analysis to figure out “how good” approximation is often much harder
- We only scratched the surface in this slide deck.