Approximation Algorithms

Admin

- Assignment 9 is due on Wednesday!
  - Questions on randomized algorithms and hashing
  - Last HW you will turn in
- Deferring lecture on randomized data structures: skip lists, cuckoo hash
- Will start approximation algorithm instead so you have some examples before you attempt HW 9 problems
- CS major advising: "drop in" hours this week (no reserved slots)
  - Can go to mine or any prof's hours and discuss plans
  - Important to fill out worksheet and submit transcript before the meeting

Approximate TSP

Traveling Salesman Problem

- Recall the traveling salesman problem: Given n cities labeled $v_1, \ldots, v_n$, and distance function $d(i, j)$, the distance from city v_i to city v_j
- **TSP. (Decision Version)** Given target D, is there a tour that visits every city and returns to the starting city with total length at most D?
- **NP complete problem.** Recall reduction from Hamiltonian cycle.
- Given directed graph $G = (V, E)$, define instance of TSP as:
  - City c_i for each node v_i
  - $d(c_i, c_j) = 1$ if $(v_i, v_j) \in E$
  - $d(c_i, c_j) = 2$ if $(v_i, v_j) \notin E$

Approximation Algorithms

- Cannot optimally solve NP hard problems like TSP efficiently
- Goal of approximation algorithms: design efficient algorithms that return solutions "close" to the optimal
- (Optimization version). Find the tour of every city with min total distance
- (Approximation algorithm for TSP).
    - Let OPT be the TSP tour of minimum total length, then a c-approximation algorithm finds a TSP tour of total length that is at most $c \cdot$ OPT, for some constant $c > 1$
    - Example: a 2-approximation for TSP always returns a tour whose cost (total distance) is never more than twice that of the optimal solution

Bad News: Approx-TSP is hard

- **Claim.** There is no polynomial-time c-approximation algorithm for the general TSP problem, for any constant $c \geq 1$, unless $\mathsf{P} = \mathsf{NP}$.
- **Proof.** Suppose there is a poly-time c-approximation algorithm A that computes a TSP tour of total weight at most $c \cdot \text{OPT}$
- Show that A can be used to solve the Hamiltonian cycle problem
- Modified reduction from Hamiltonian cycle instance G to TSP instance:
  - $d(c_i, c_j) = 1$ if $(v_i, v_j) \in E$ and $d(c_i, c_j) = cn + 1$ if $(v_i, v_j) \notin E$
- If G has a Hamiltonian cycle: there is a tour of length exactly n
- If G does not have a Hamiltonian cycle, any tour has length at least $cn + 1$

Bad News: Approx-TSP is hard

- **Claim.** There is no polynomial-time c-approximation algorithm for the general TSP problem, for any constant $c \geq 1$, unless $\mathsf{P} = \mathsf{NP}$.
- **Proof. (Cont).** If G has a Hamiltonian cycle: there is a tour of length exactly n
- If G does not have a Hamiltonian cycle, any tour has length at least $cn + 1$
- A computes tour of length at most $cn \iff G$ has a Hamiltonian cycle:
    - A solves Hamiltonian cycle in polynomial time and $\mathsf{P} = \mathsf{NP}$
- **[More Bad news]**
  For any function $f(n)$ that can be computed in polynomial time in n, there is no polynomial-time $f(n)$-approximation for TSP on general weighted graphs, unless $\mathsf{P} = \mathsf{NP}$.

Good News: Metric TSP is Not

- While approximating TSP on general distances is NP hard, the common special case can be approximate easily
- **Metric TSP.** TSP problem on metric distances, that is, d satisfies:
    - $d(i,j) \leq d(i,k) + d(k,j)$ for any cities i, j, k [Triangle inequality]
    - $d(i,i) = 0$ and $d(i,j) \geq 0$ [Identity and Non-negative]
    - $d(i,j) = d(j,i)$ [Symmetric]
- Eucliean distances are an example of metric distances
- **Metric TSP** is still NP complete (reduction from undirected Ham cycle)
    - Setting $d(c_i, c_j) = 2$ when $(v_i, v_j) \notin E$ satisfies triangle inequality

Approximating Metric TSP

Approximating an NP Hard Problem

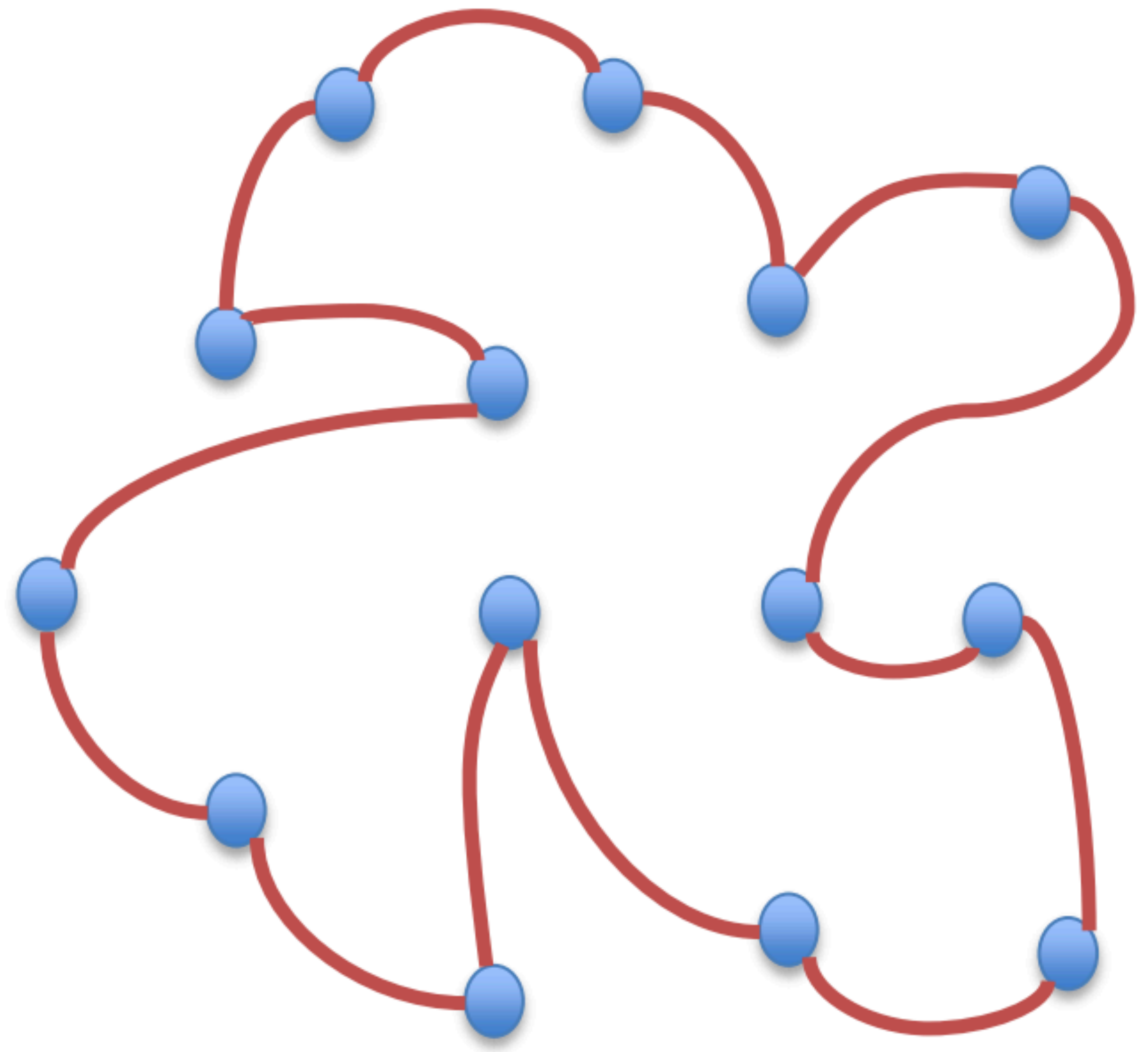
- (Metric TSP) Consider the **weighted complete graph** G where each vertex is a city, and each edge (i, j) for $i, j \in V$ has weight equal to the distance $d(i, j)$, where d satisfies the triangle inequality
- (NP hard problem). Find the tour of every city with min total distance
- (Goal: c-approximation). Design an algorithm that finds a TSP tour of total length that is at most $c \cdot$ OPT, for some constant $c > 1$
- Remember, we don't actually know what the optimal algorithm or OPT actually is: we need to approximate without knowing that
- We do this by relating (upper and lower bounding) the cost of the approximation algorithm and OPT via a carefully chosen function

Approximating Metric TSP

- Consider the weighted complete graph G where each vertex is a city, and each edge (i, j) for $i, j \in V$ has weight equal to the distance $d(i, j)$, where d satisfies the triangle inequality
- (Optimization version). Find the tour of every city with min total distance
- Steps to follow when designing an approximation algorithm for a minimization problem
    - **Lower bound the optimal cost** by some function of input
    - Upper bound the cost of algorithm by **the same function**
- Will use MSTs to derive these upper and lower bounds for metric TSP
- We give a 2-approximation to metric-TSP using minimum spanning trees

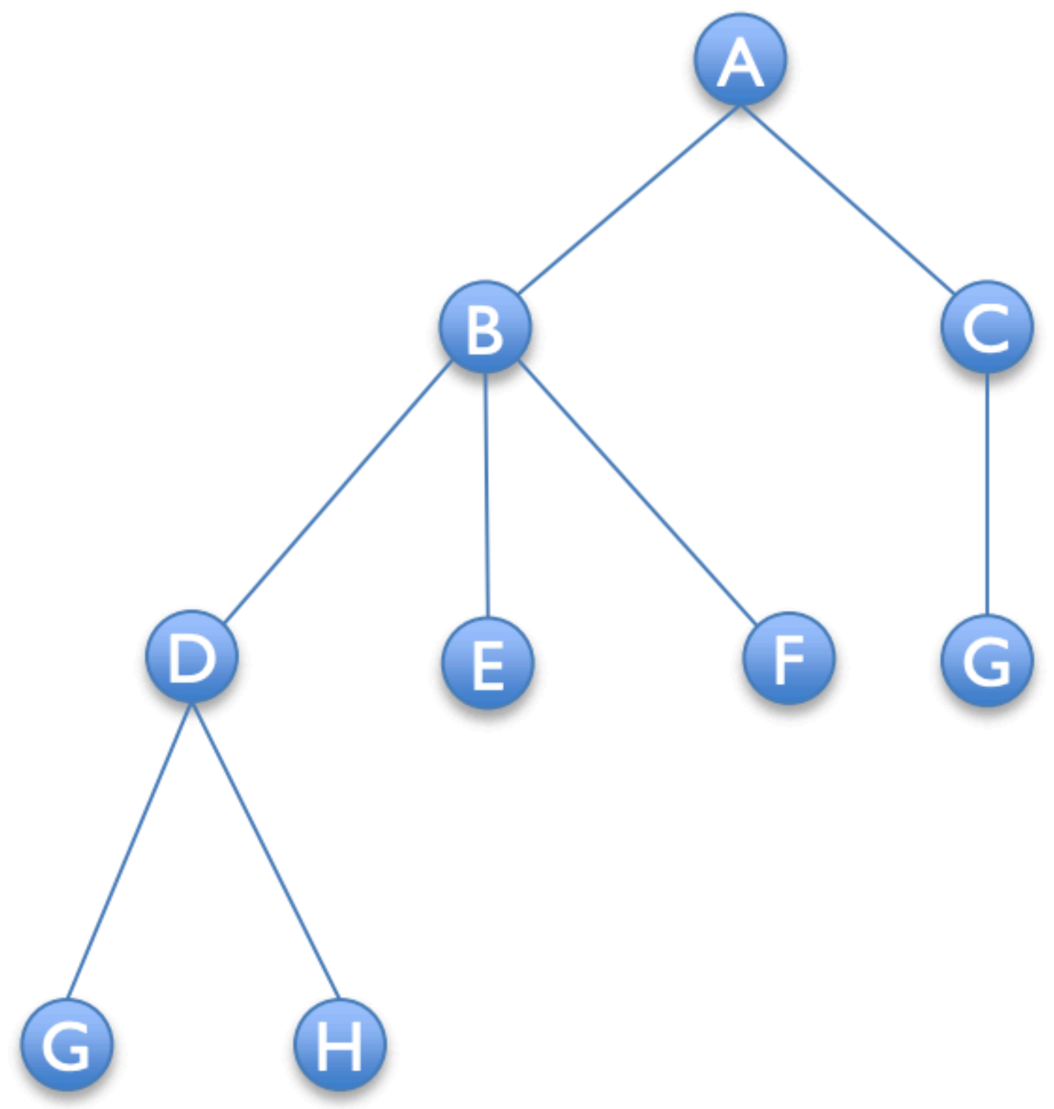
Lower Bound on OPT

- Note that an optimal tour must not visit a city more than once:
  - Optimal tour is a simple cycle that visits all nodes
- **Claim**. Let T be the minimum spanning tree of G then length of the optimal tour OPT $\geq w(T)$.
- **Proof**. Take an optimal tour of length OPT
- Drop an edge from it to obtain a spanning tree T'
- Distances/weights are non-negative, so $w(T') \leq$ OPT
- $w(T) \leq w(T')$ (T is the MST)
- Thus $w(T) \leq$ OPT ∎



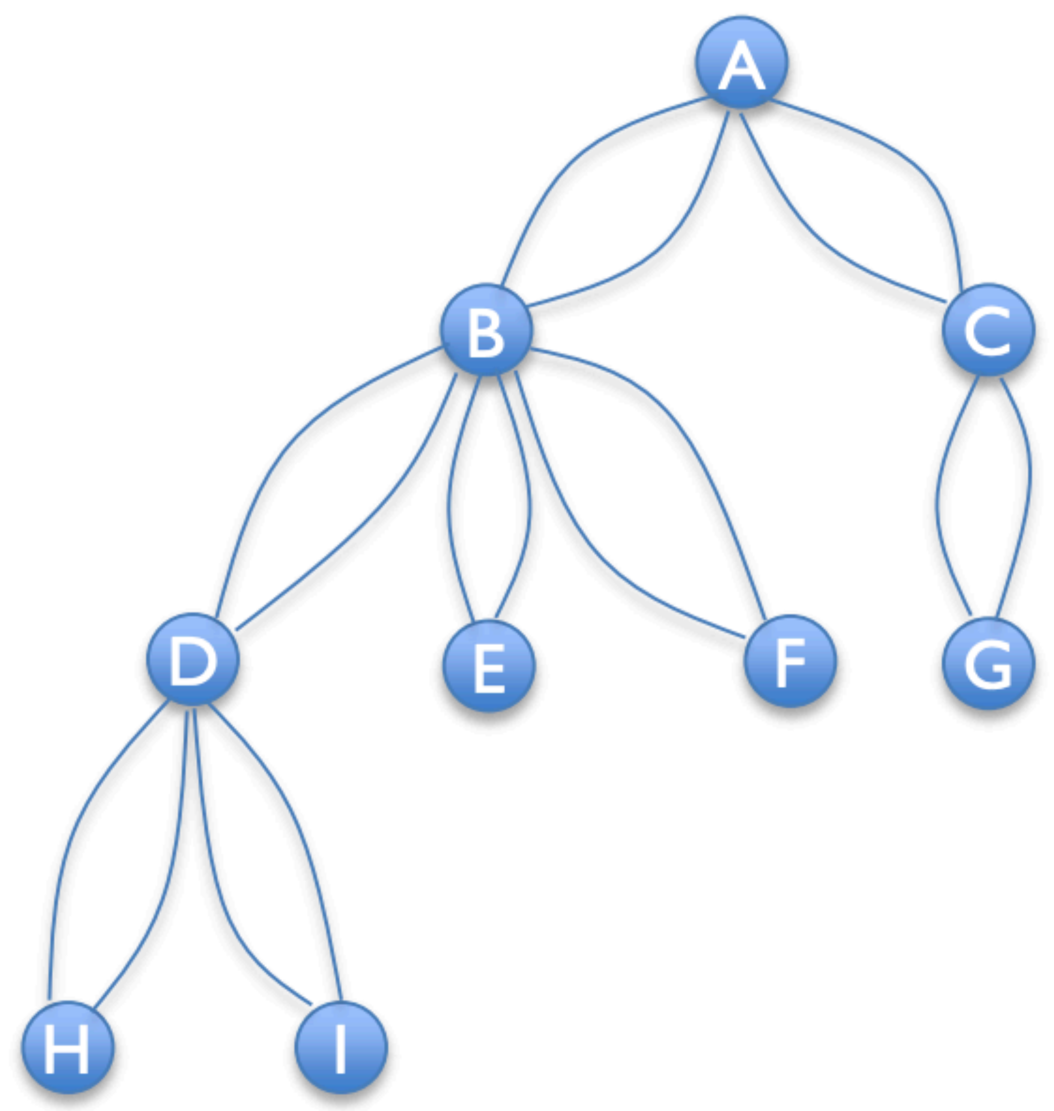
Double Tree Algorithm

- **Find a minimum spanning tree T**
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- Shortcut Euler tour to avoid repeated vertices



Double Tree Algorithm

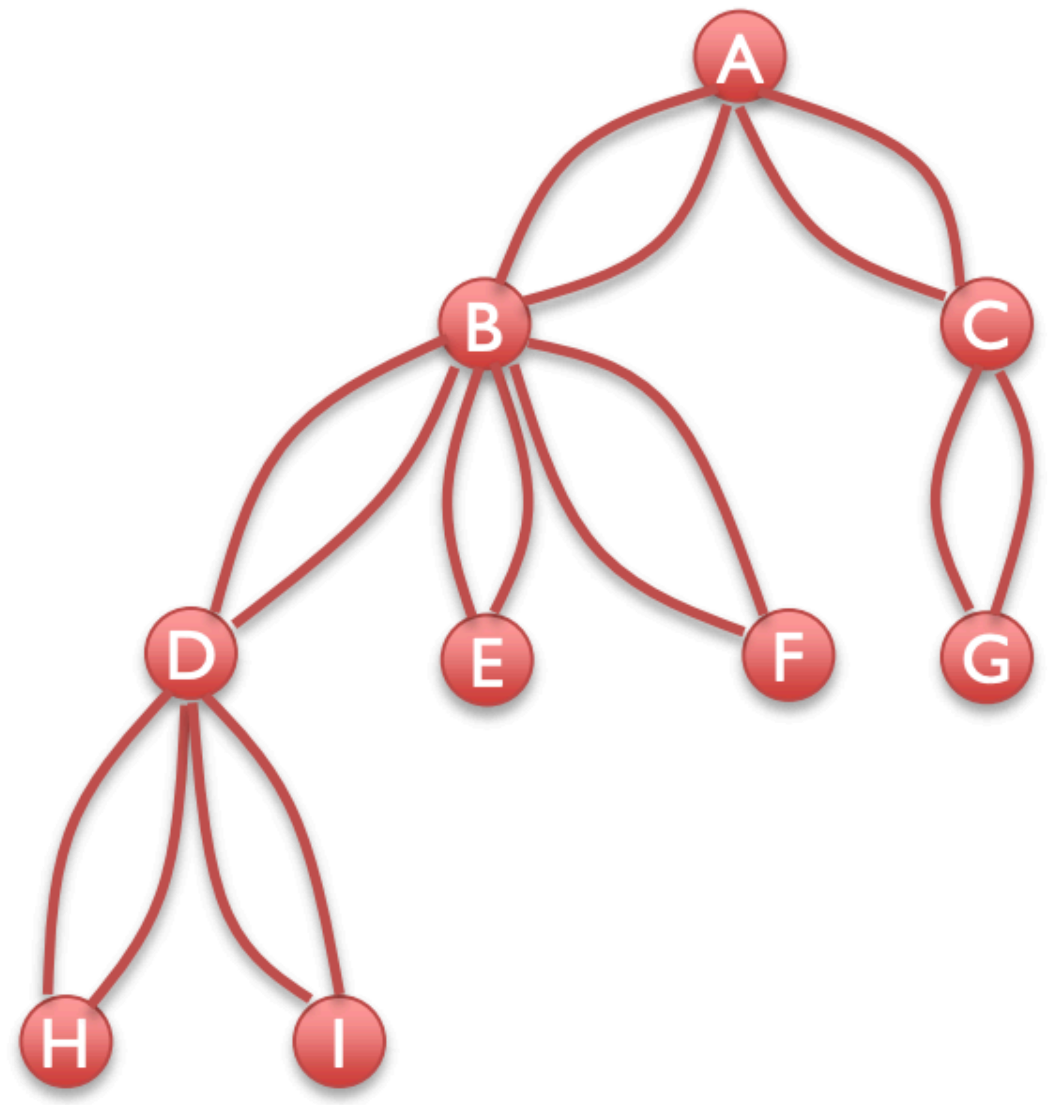
- Find a minimum spanning tree T
- **Duplicate every edge in** T
- Find an Eulerian tour of resulting multi-graph
- Shortcut Euler tour to avoid repeated vertices



Double Tree Algorithm

- Find a minimum spanning tree T
- Duplicate every edge in T
- **Find an Eulerian tour** of resulting multi-graph
- Shortcut Euler tour to avoid repeated vertices

Why must an Euler tour exist?

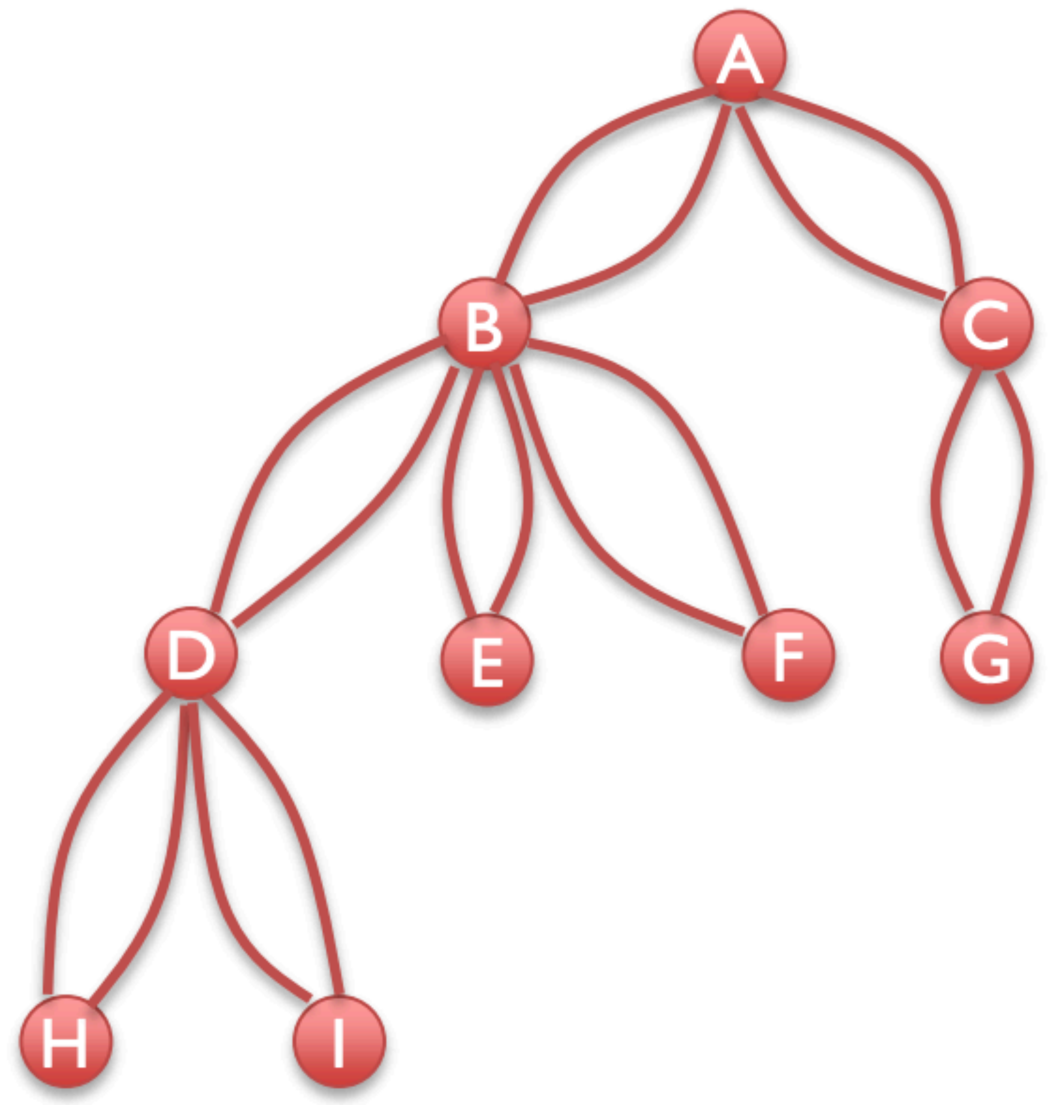


A,B,D,H,D,I,D,B,E,B,F,B,A,C,G,C,A

Double Tree Algorithm

- Find a minimum spanning tree T
- Duplicate every edge in T
- **Find an Eulerian tour** of resulting multi-graph
- Shortcut Euler tour to avoid repeated vertices

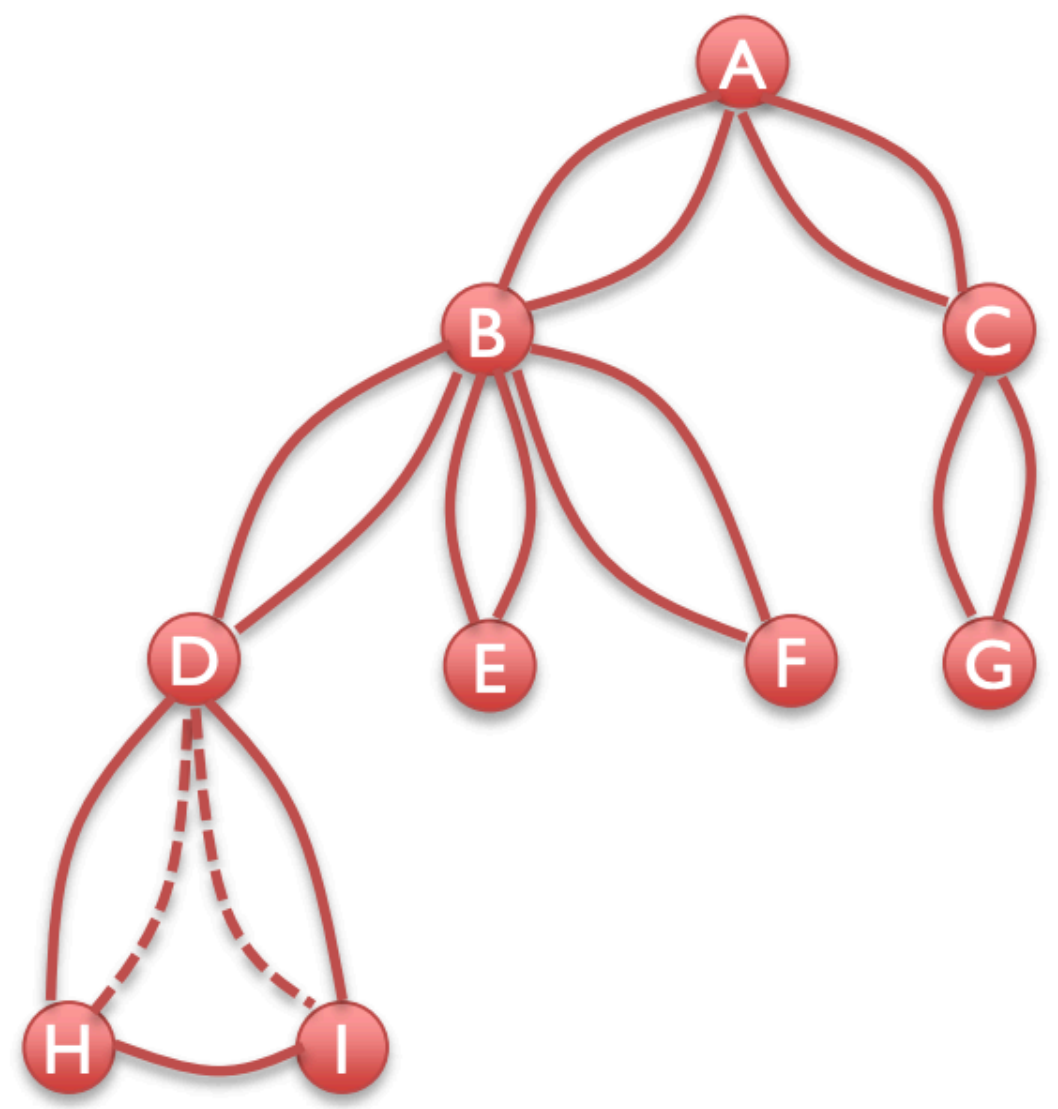
A graph has an Euler tour iff all nodes have even degree



A,B,D,H,D,I,D,B,E,B,F,B,A,C,G,C,A

Double Tree Algorithm

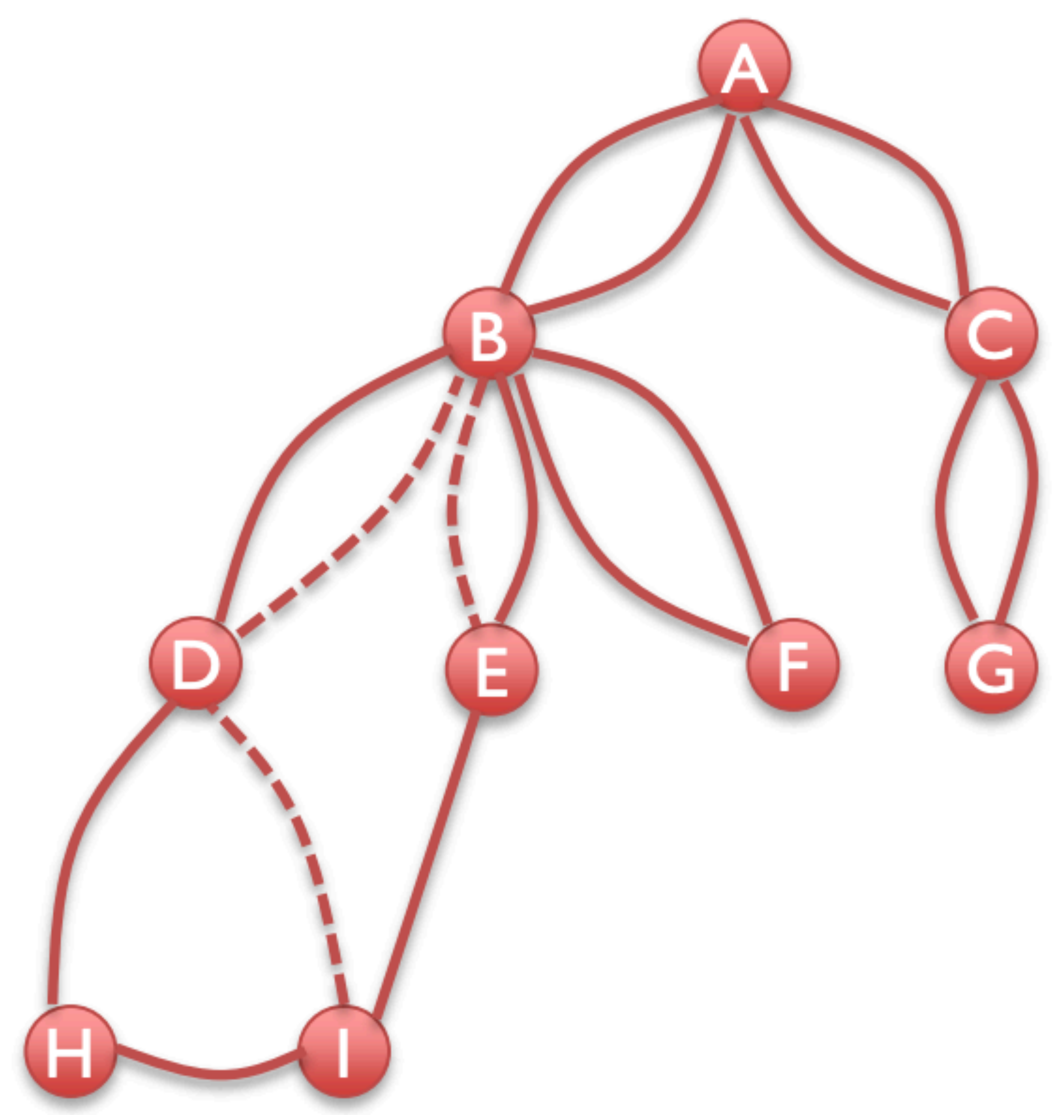
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



A,B,D,H,~~D~~,I,D,B,E,B,F,B,A,C,G,C,A

Double Tree Algorithm

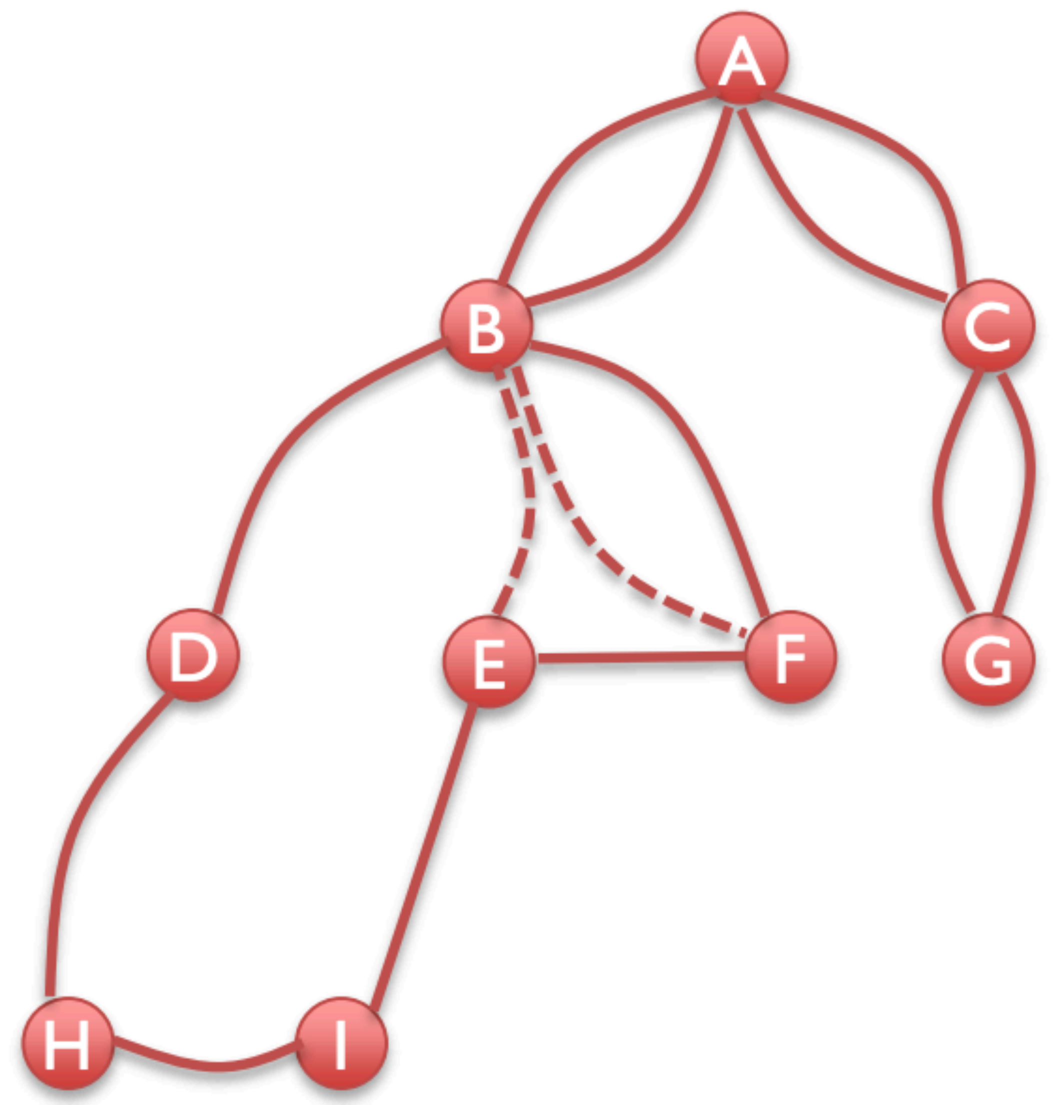
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



A,B,D,H,$\xcancel{D}$,I,$\xcancel{D}$,$\xcancel{B}$,E,B,F,B,A,C,G,C,A

Double Tree Algorithm

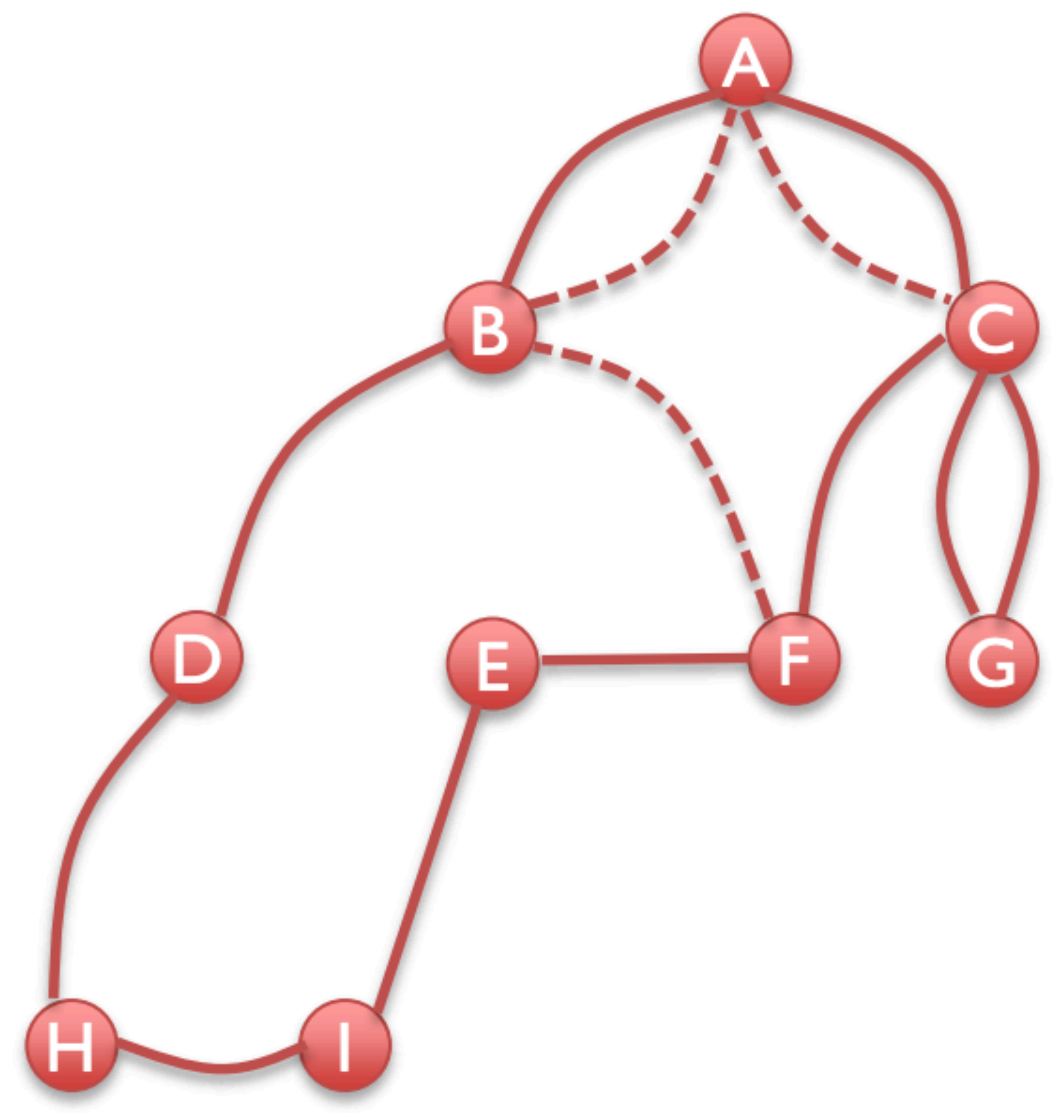
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



A,B,D,H,$\xcancel{D}$,I,$\xcancel{D}$,$\xcancel{B}$,E,$\xcancel{B}$,F,B,A,C,G,C,A

Double Tree Algorithm

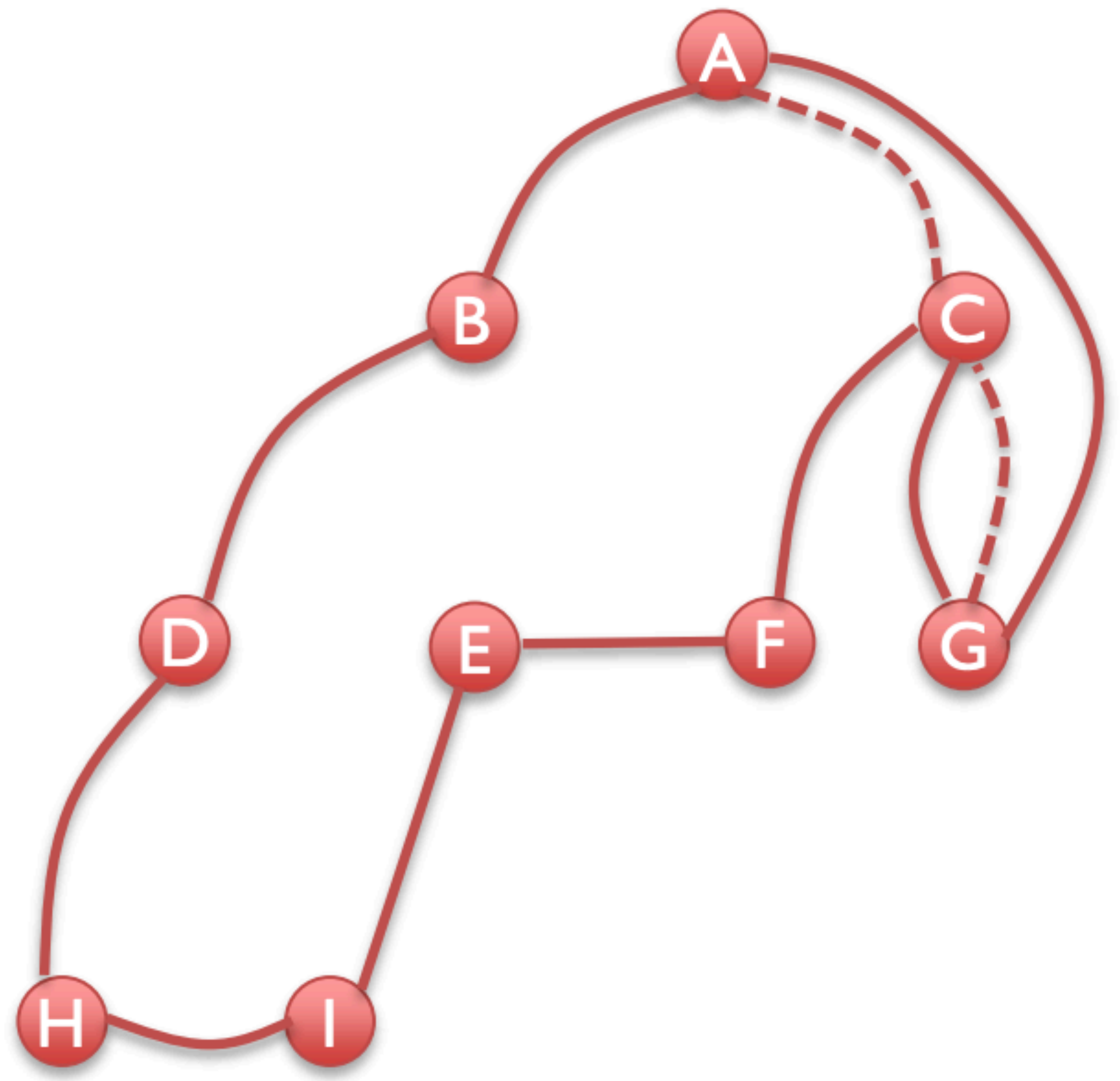
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



A,B,D,H,~~D~~,I,~~D~~,~~B~~,E,~~B~~,F,~~B~~,~~A~~,C,G,C,A

Double Tree Algorithm

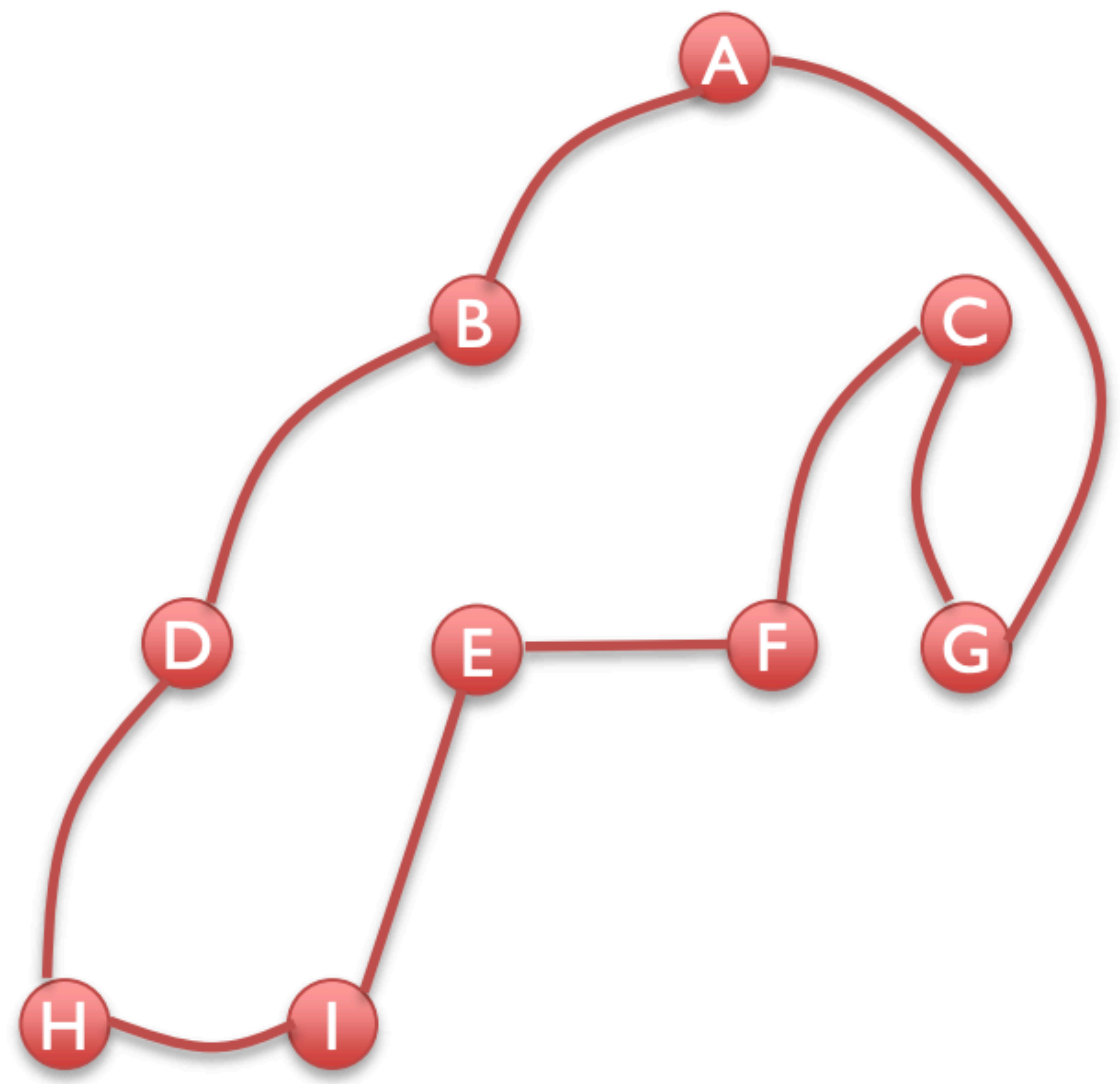
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



A,B,D,H,~~D~~,I,~~D~~,~~B~~,E,~~B~~,F,~~B~~,~~A~~,C,G,~~C~~,A

Double Tree Algorithm

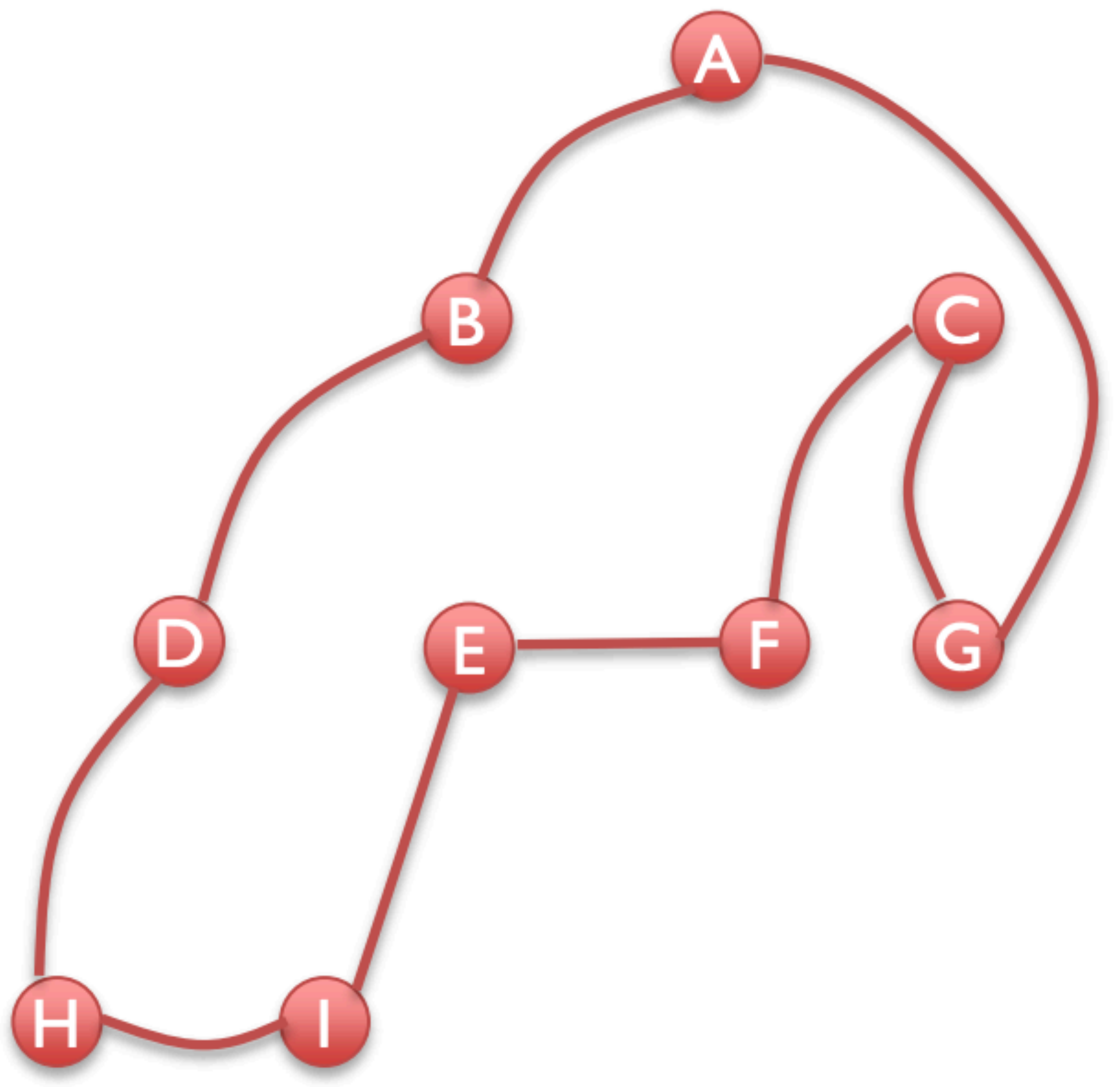
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



A,B,D,H,~~D~~,I,~~D~~,~~B~~,E,~~B~~,F,~~B~~,~~A~~,C,G,~~C~~,A

Double Tree Analysis

- **Claim.** Double-tree algorithm is a 2-approximation to TSP.
- **Proof.** The Euler tour visits every edge of MST T exactly twice, thus the length of tour $= 2 \cdot w(T)$
- Due to triangle inequality, shortcutting the tour does not increase length
    - (Here is where metric distances play a role!)
- Thus length of short-circuited tour $\leq 2 \cdot w(T)$
- Since $w(T) \leq$ OPT, we get that our tour length is $\leq 2 \cdot$ OPT ∎



$A,B,D,H,\cancel{D},I,\cancel{D},\cancel{B},E,\cancel{B},F,\cancel{B},\cancel{A},C,G,\cancel{C},A$

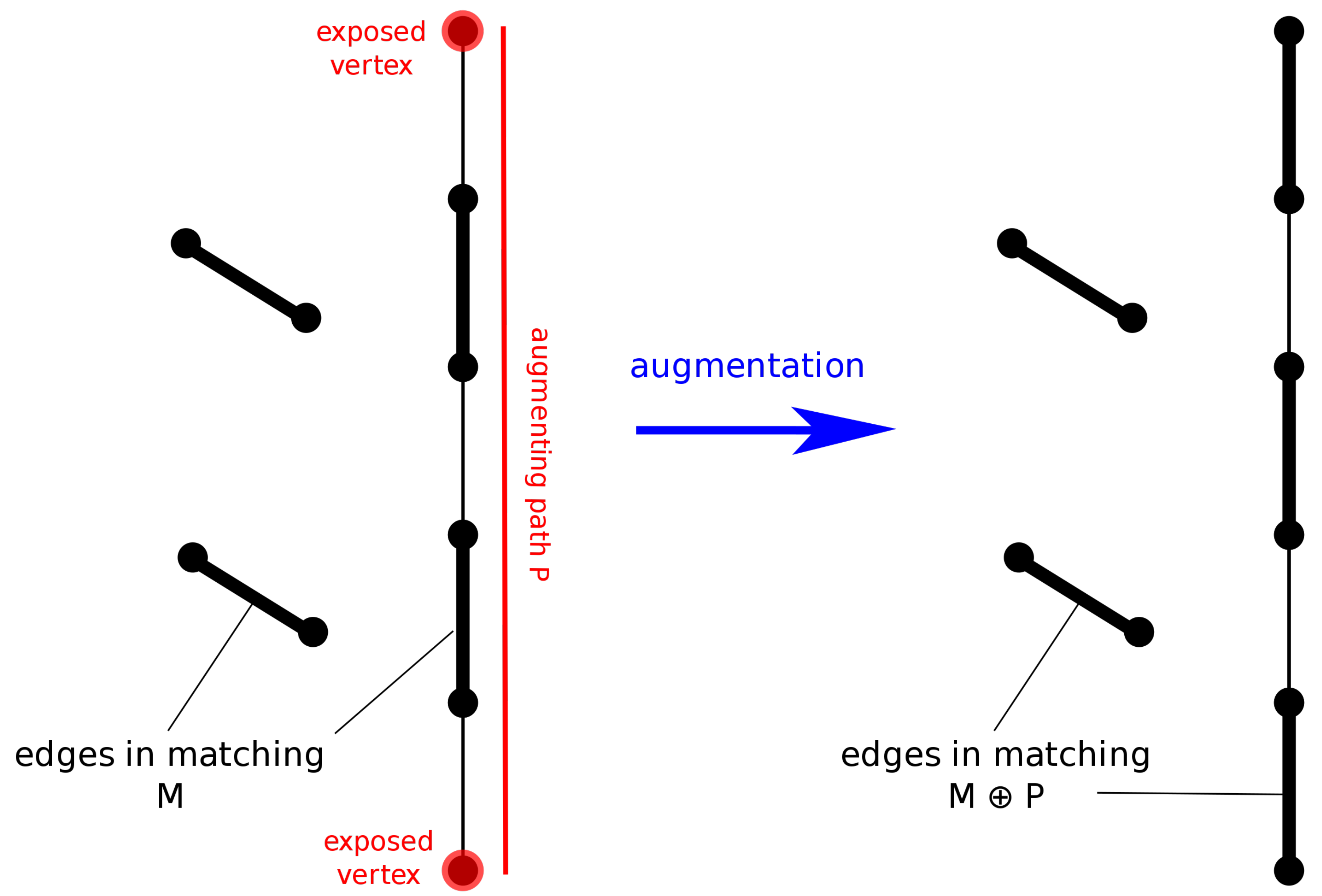
Christofides-Serdyukov Algorithm

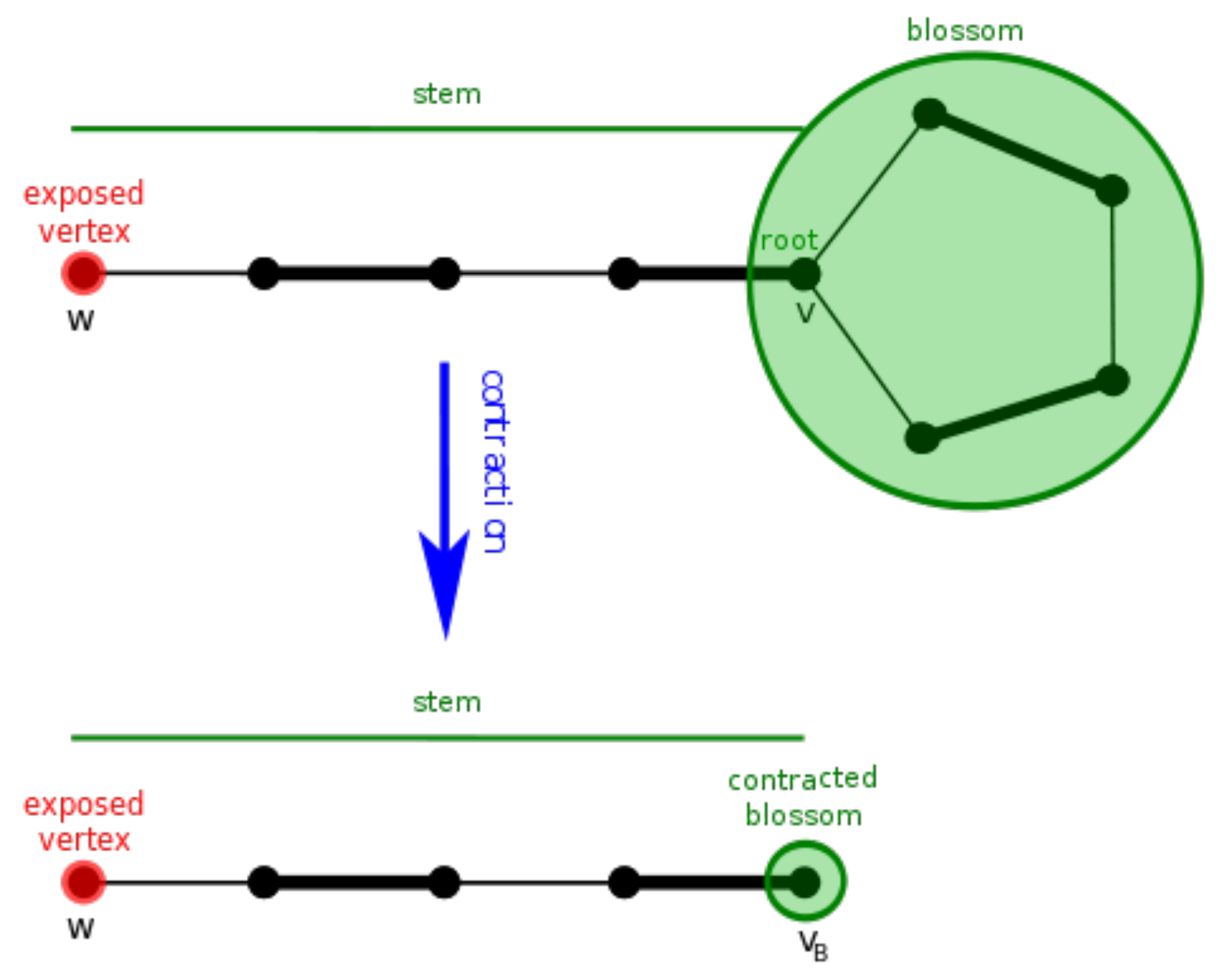
Christofides Algorithm [Christofides 76][Serdyukov 76]

- Doubling the edges of MST is one way to achieve even degree nodes, but is there a cheaper way to augment to tree to obtain an Eulerian tour?
- What is the parity of odd degree vertices in an undirected graph?
    - Even number of odd degree vertices!
- **Christofides algorithm.** Starts with an MST, but fixes the parity of odd degree vertices by augmenting it with a matching
- **Matching**. A set of edges such that no two are adjacent
- **Perfect matching**. Every vertex is incident to exactly one edge in the matching
- **Fact We'll Use.** Minimum cost "perfect" matchings of any graph can be computed in polynomial time.

Minimum Cost Matching

- Won't see in this class, unfortunately
- Edmond's "blossom" algorithm
- $O(|E||V|^2)$ (slow, but much better than exponential)
- Somewhat similar to Ford-Fulkerson:
  - Use special structure to prove that we just need to find augmenting paths
  - Use data structures so that we can find augmenting paths quickly
- Tricky part: "augmenting paths" are more complicated when finding a matching



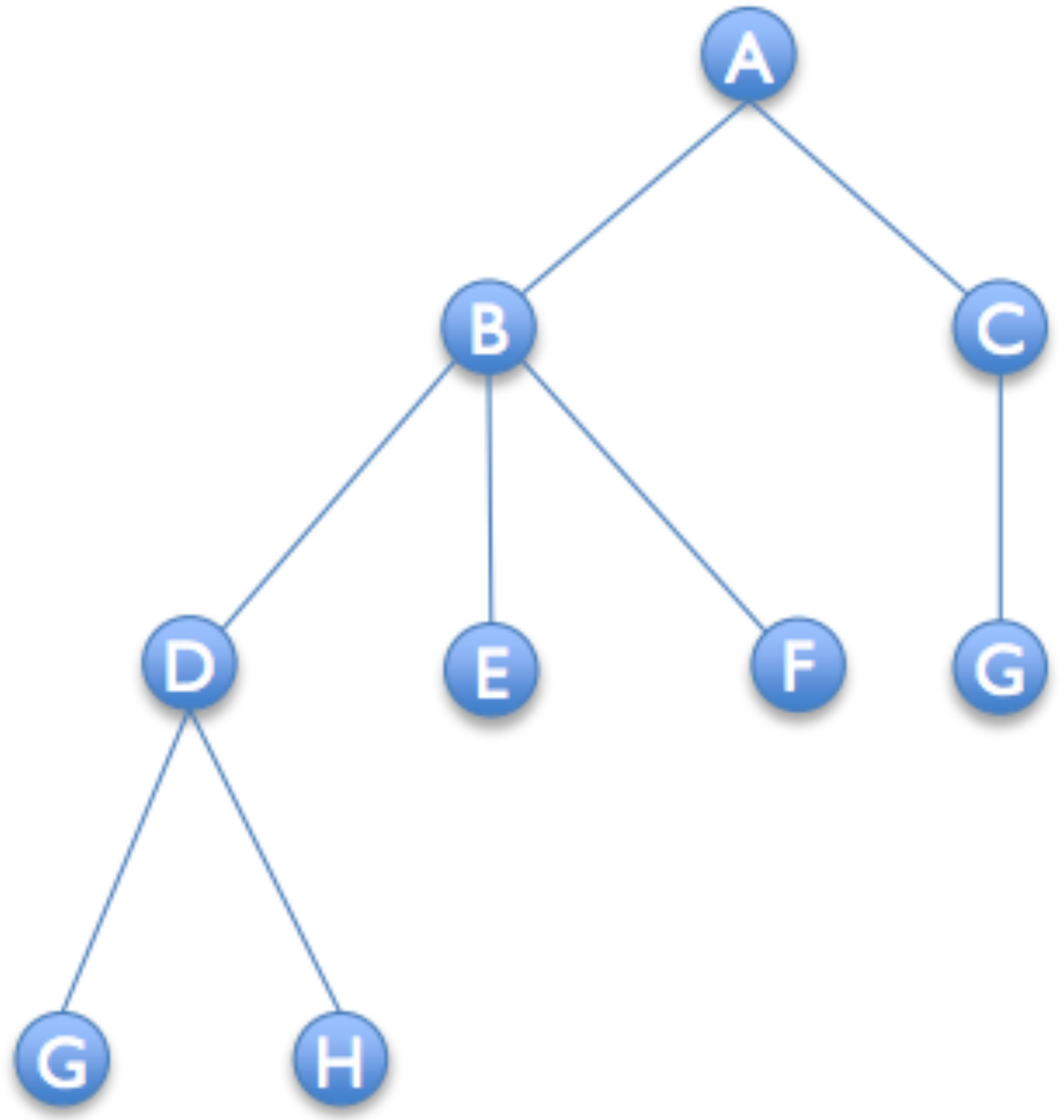


Christofides Algorithm

- Find the minimum spanning tree T
- Compute O: the set of odd degree vertices in T
- Find the min-cost perfect matching M of subgraph induced by O
- Return shortcut of Euler tour of $T \cup M$

$|O|$ must be even

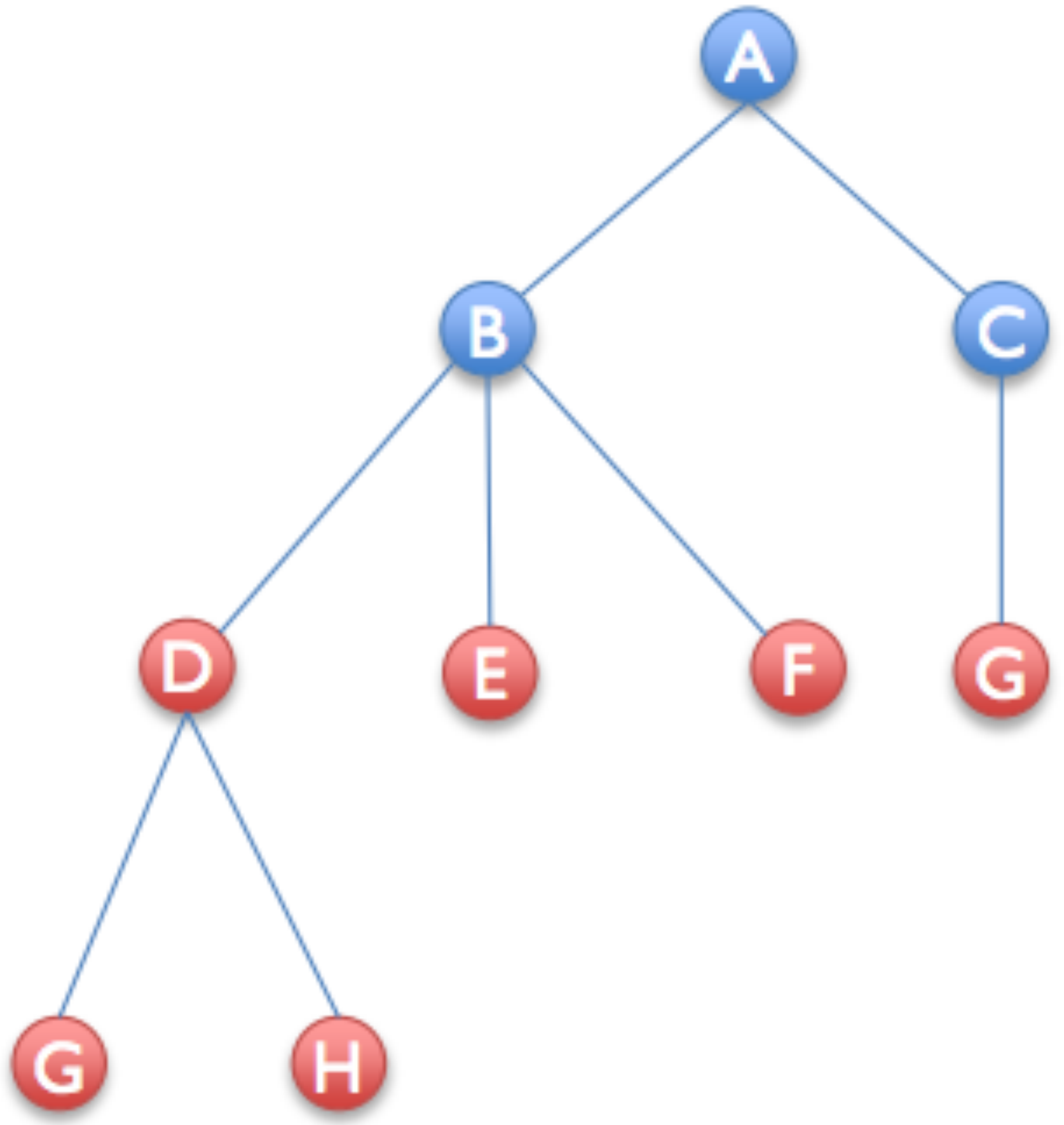
All odd-degree vertices and any edges connecting them



Christofides Algorithm

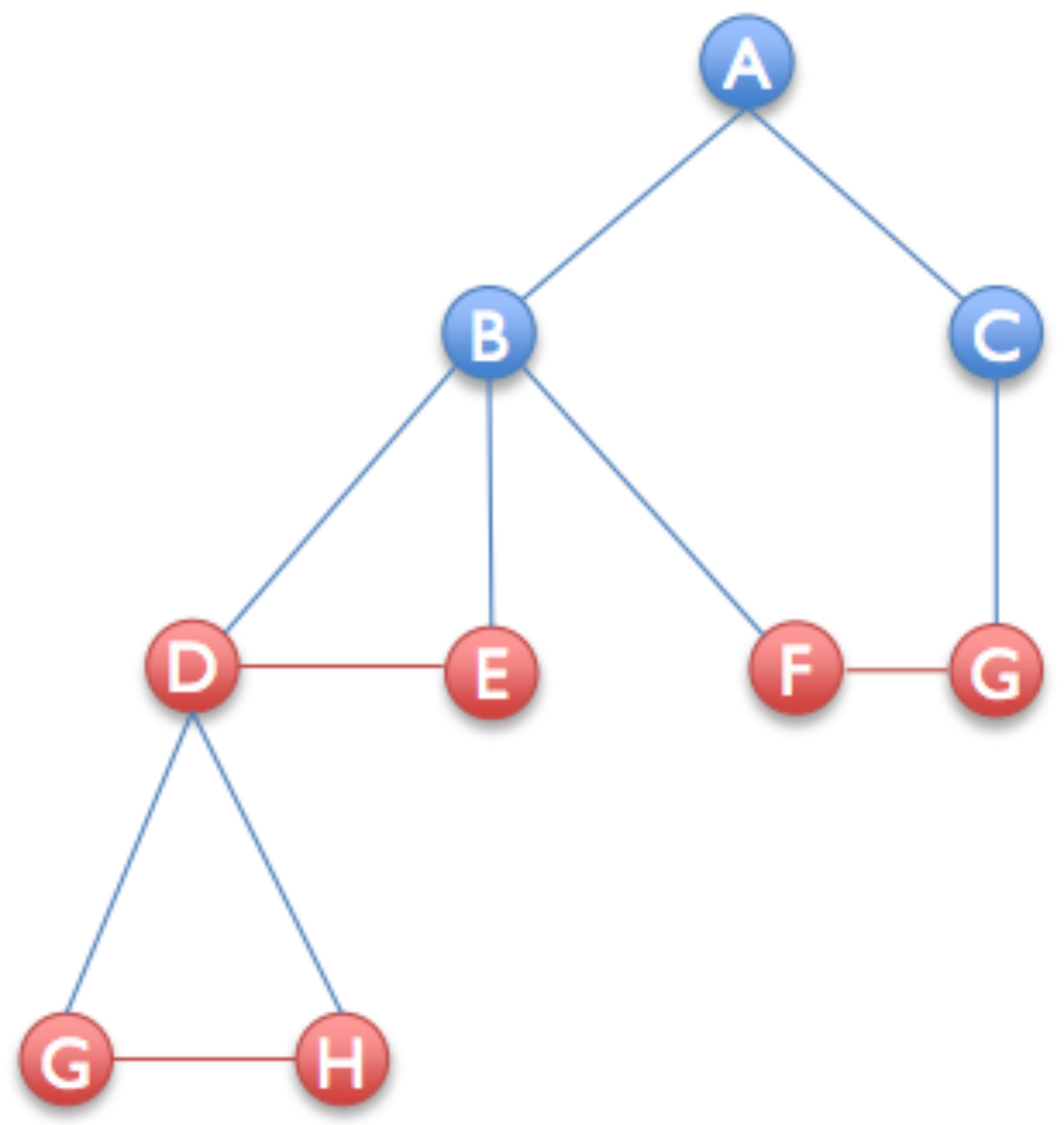
- Find the minimum spanning tree T
- Compute O: the set of odd degree vertices in T
- Find the min-cost perfect matching M of subgraph induced by O
- Return shortcut of Euler tour of $T \cup M$

What does adding M do to O?



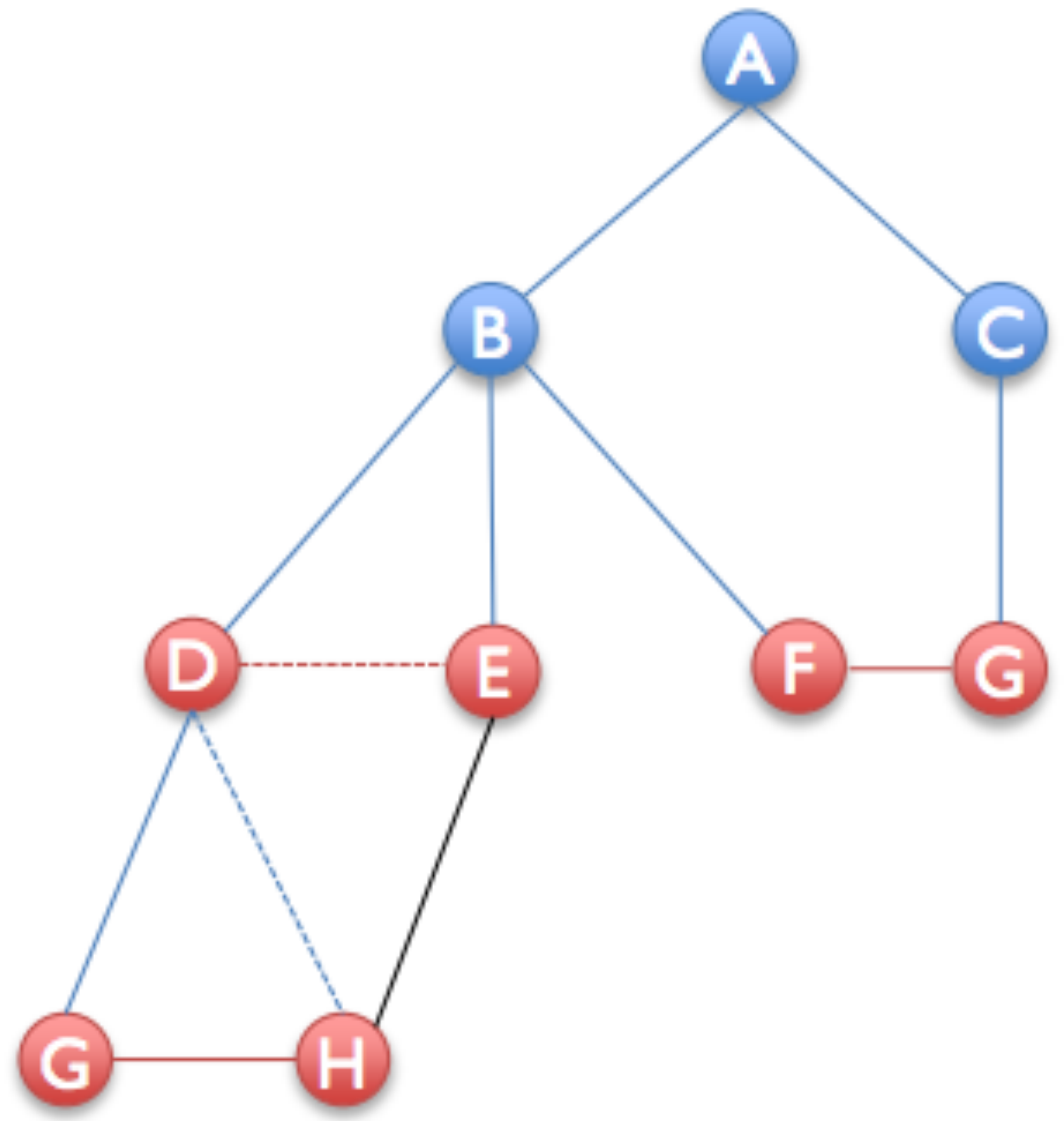
Christofides Algorithm

- Find the minimum spanning tree T
- Compute O: the set of odd degree vertices in T
- Find the min-cost perfect matching M of subgraph induced by O
- Return shortcut of Euler tour of $T \cup M$



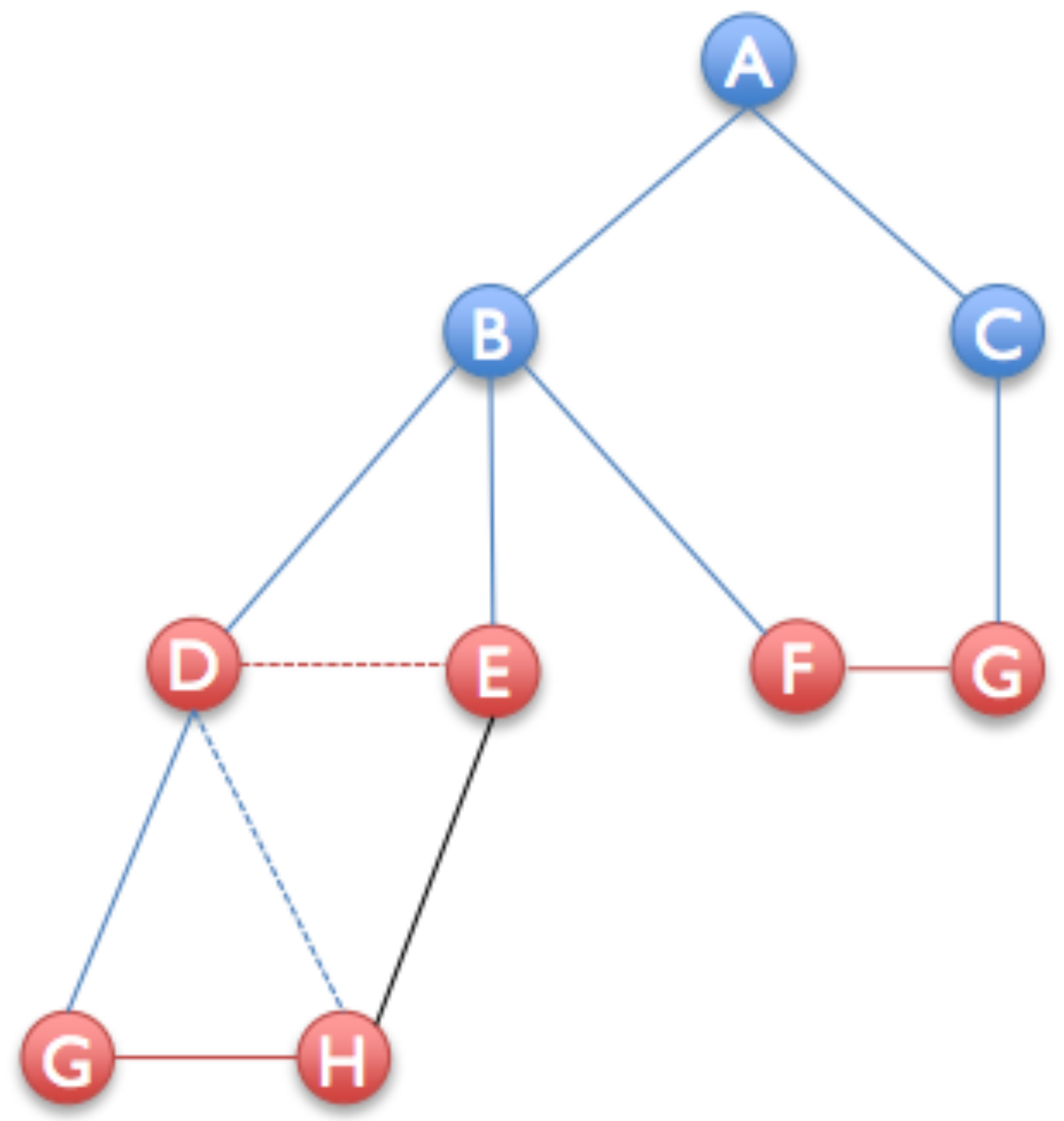
Christofides Algorithm

- Find the minimum spanning tree T
- Compute O: the set of odd degree vertices in T
- Find the min-cost perfect matching M of subgraph induced by O
- Return shortcut of Euler tour of $T \cup M$



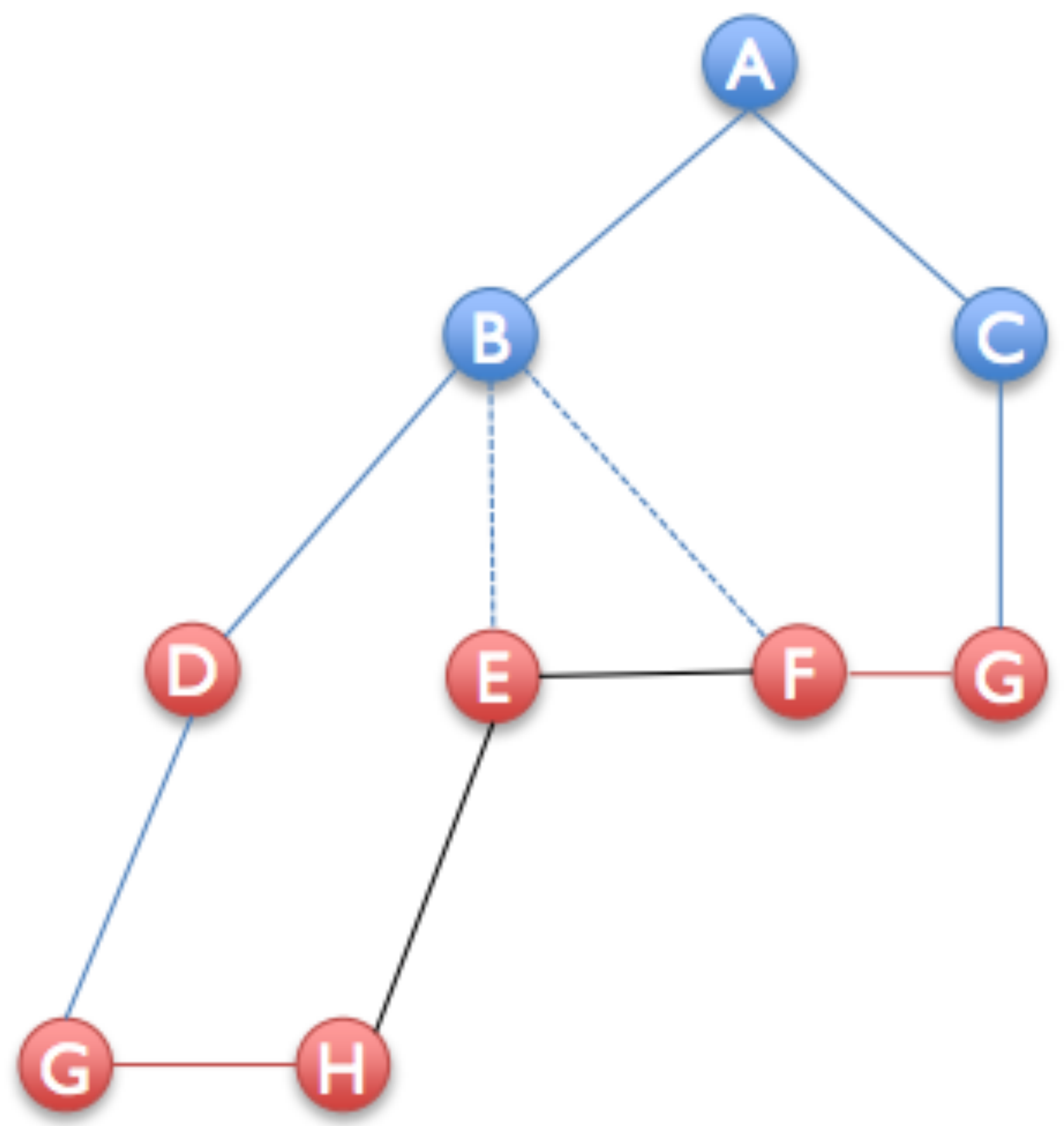
Christofides Algorithm

- Find the minimum spanning tree T
- Compute O: the set of odd degree vertices in T
- Find the min-cost perfect matching M of subgraph induced by O
- Return shortcut of Euler tour of $T \cup M$



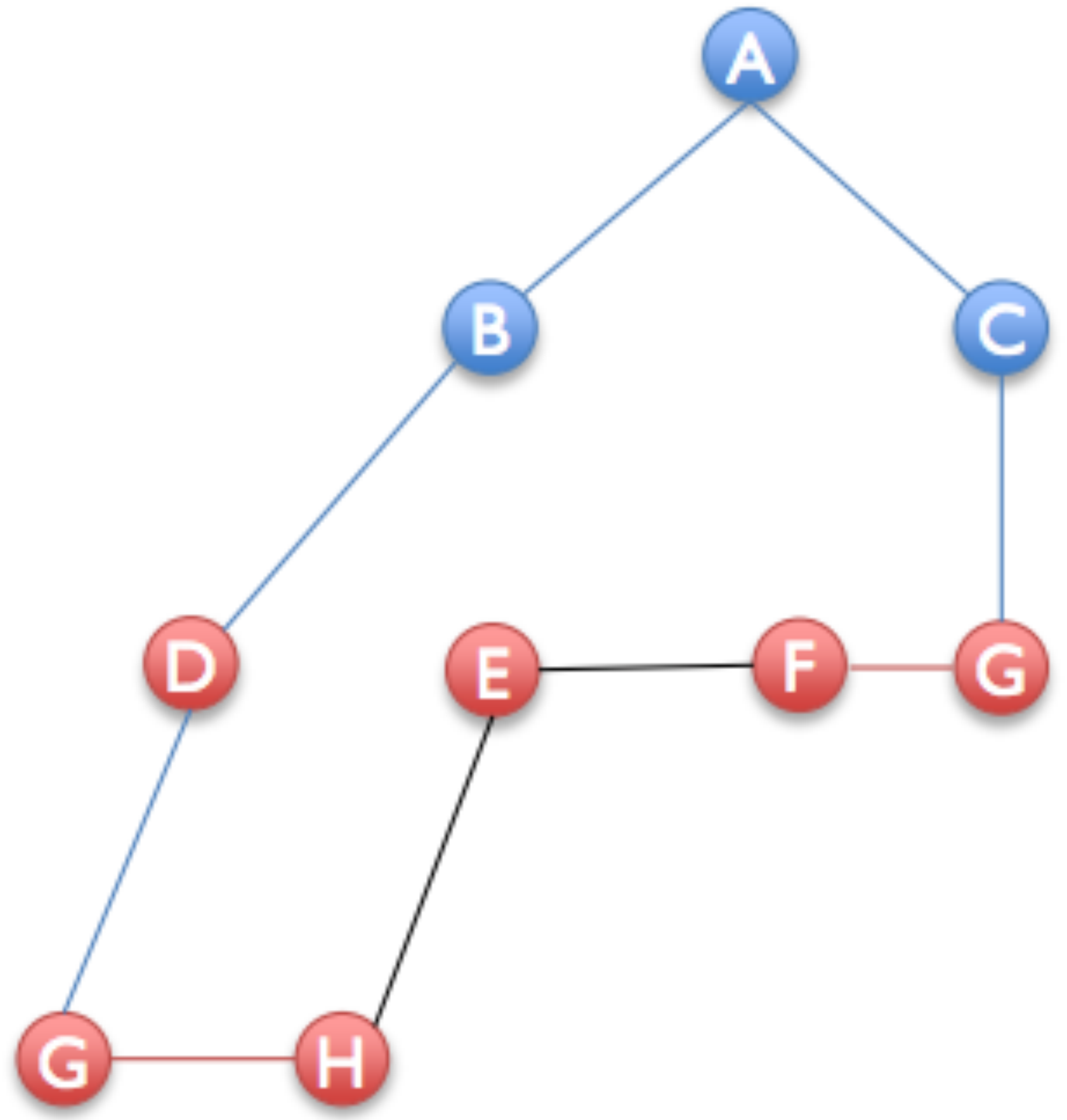
Christofides Algorithm

- Find the minimum spanning tree T
- Compute O: the set of odd degree vertices in T
- Find the min-cost perfect matching M of subgraph induced by O
- Return shortcut of Euler tour of $T \cup M$



Christofides Algorithm

- Find the minimum spanning tree T
- Compute O: the set of odd degree vertices in T
- Find the min-cost perfect matching M of subgraph induced by O
- Return shortcut of Euler tour of $T \cup M$



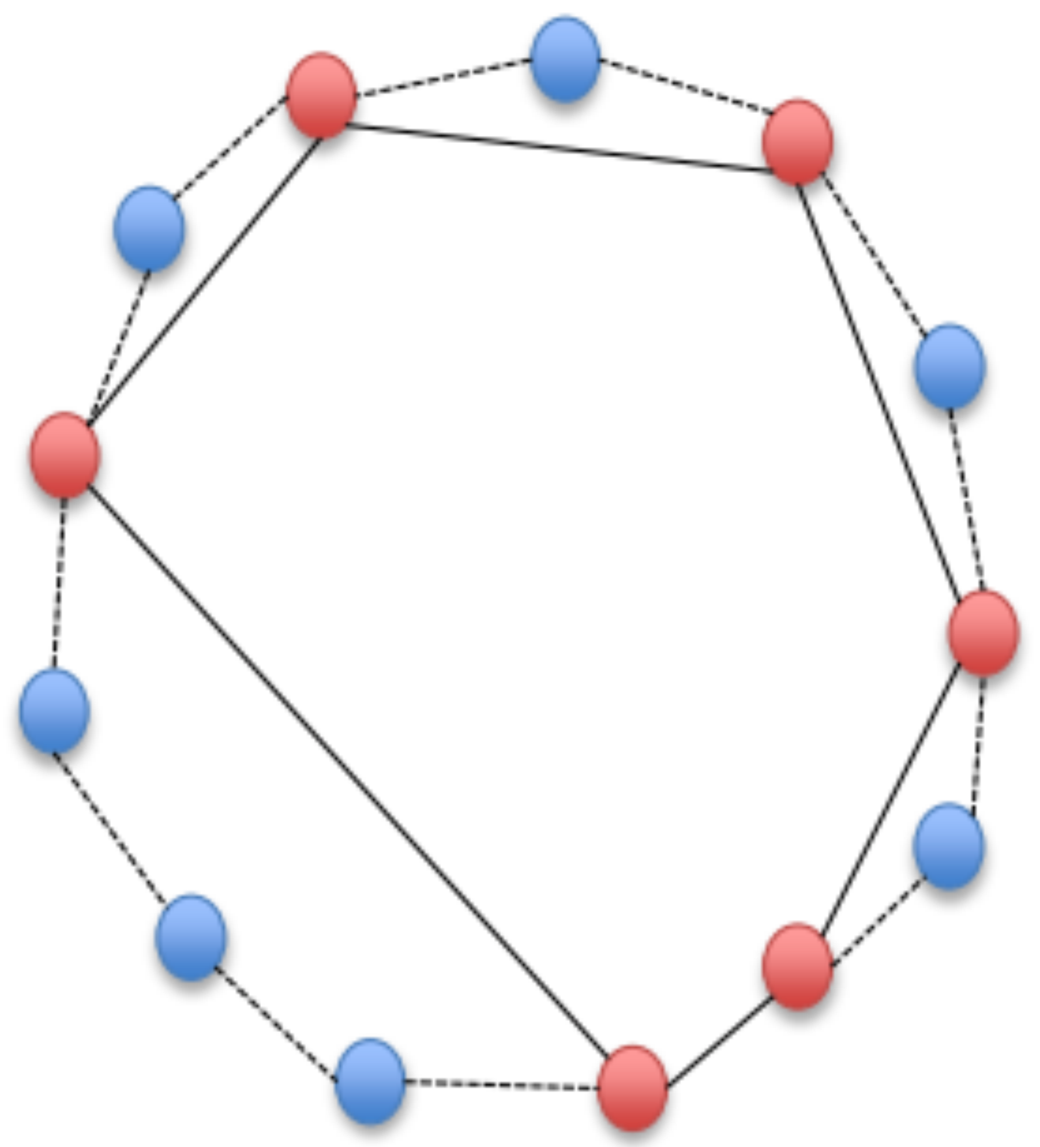
Christofides Analysis

- Cost of TSP tour returned is at most $w(T) + w(M)$
  - T is the MST of the graph, and M is the minimum cost perfect matching on the subgraph induced by O (odd degree nodes in T)
- We know OPT $\geq w(T)$
- To relate the costs, we lower bound the OPT in terms of the cost of M
- **Claim**. Let OPT be the length of the optimal tour and let M be a minimum-cost perfect matching on the complete subgraph induced by O, the odd degree nodes in MST T, then

$$w(M) = \sum_{e \in M} w_e \leq \frac{1}{2} \cdot \text{OPT}$$

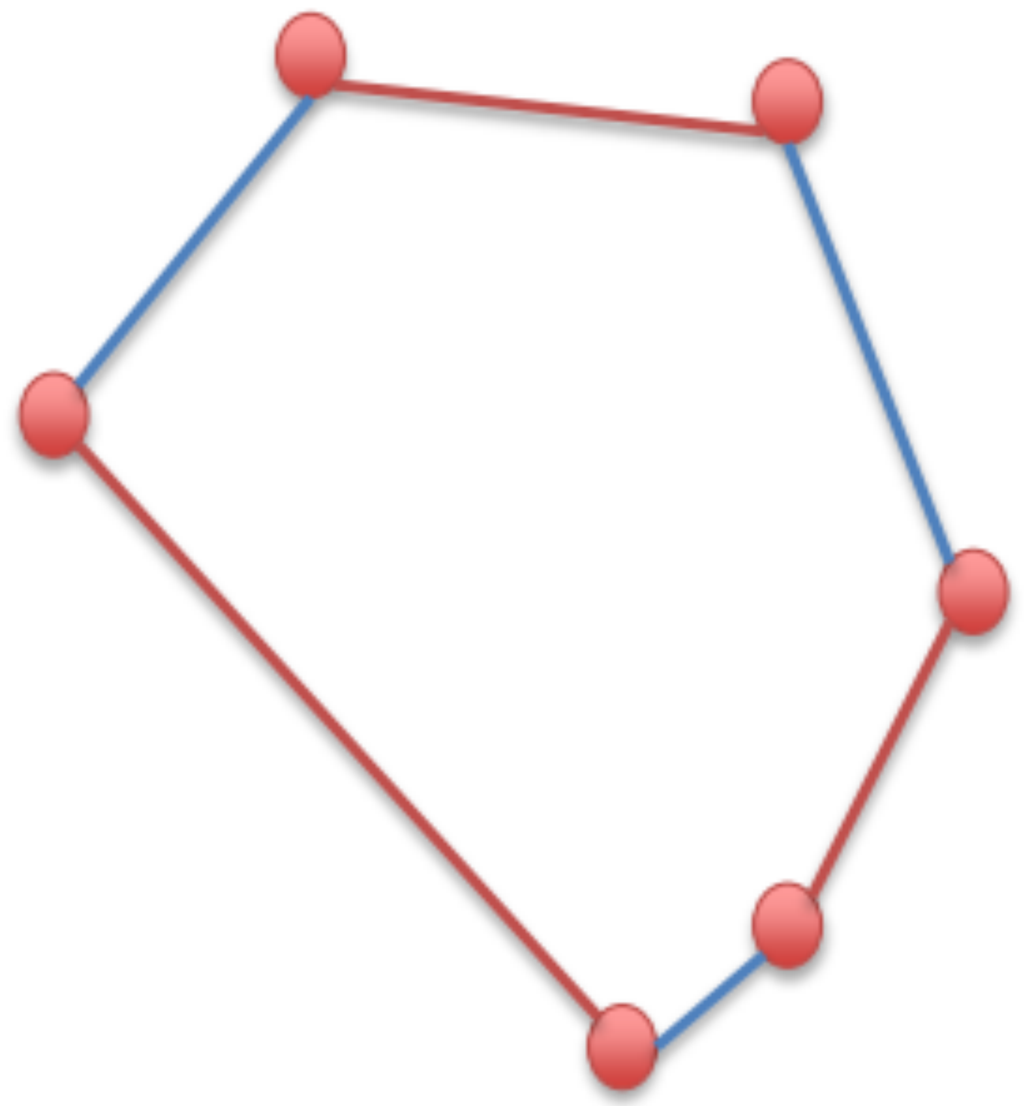
Christofides Analysis

- **Proof of claim.** Consider an optimal tour with cost OPT and consider vertices in O, the odd-degree vertices in T
- Shortcut optimal tour to obtain tour of vertices in O
- By triangle inequality the cost of tour can only decrease



Christofides Analysis

- **Proof of claim.** Consider an optimal tour with cost OPT and consider vertices in O, the odd-degree vertices in T
- Shortcut optimal tour to obtain tour of vertices in O
- By triangle inequality the cost of tour can only decrease
- Consider matchings M_1, M_2 created by alternating edges on this tour
- $w(M_1) + w(M_2) \leq$ OPT
- Then, $\min\{w(M_1), w(M_2)\} \leq$ OPT$/2$
- $w(M) \leq \min\{w(M_1, w(M_2)\}$, where M:min-cost perfect matching on subgraph induced by O
- Thus, $w(M) \leq$ OPT$/2$

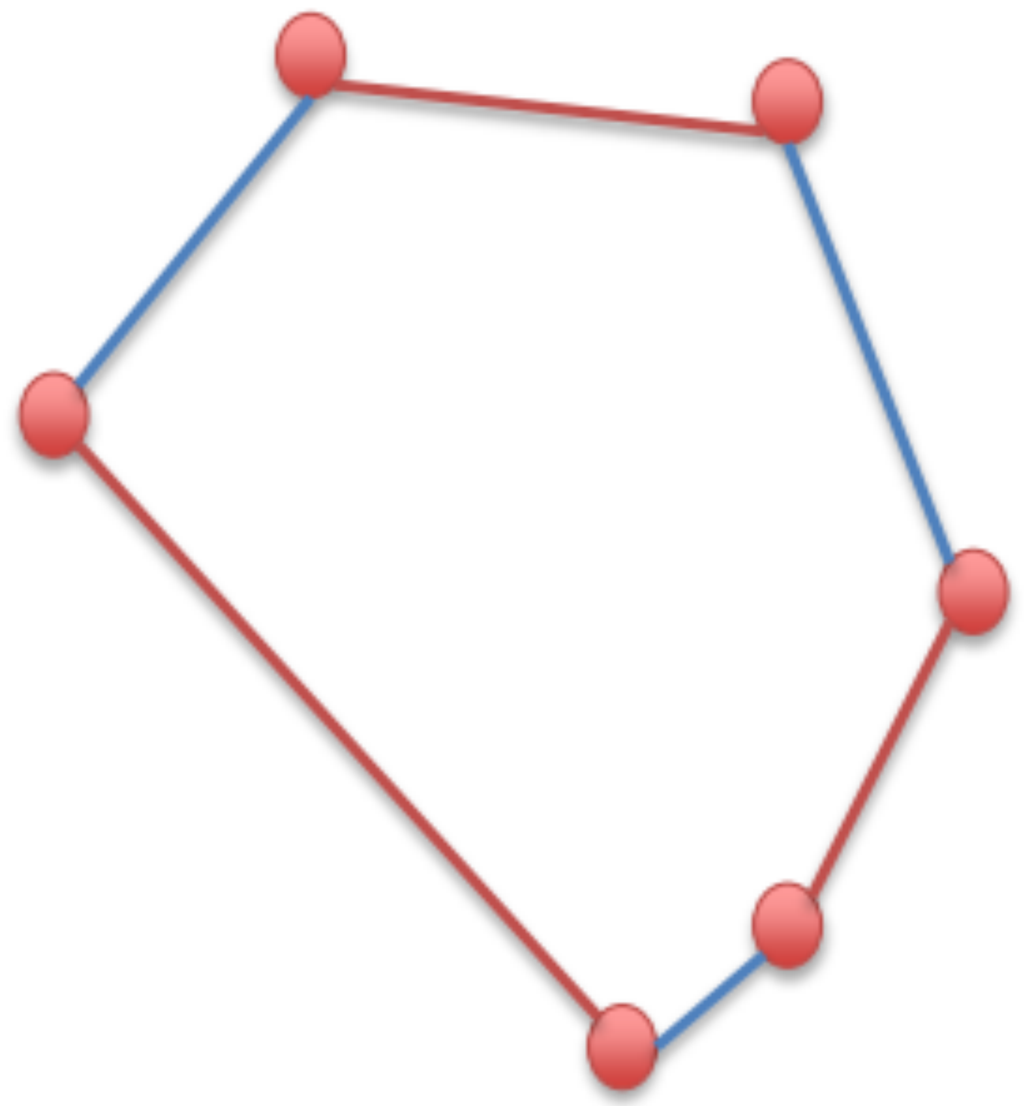


Wrapping Up

- Cost of TSP tour returned by Christofides $\leq w(T) + w(M)$
- We showed that OPT (optimal cost) $\geq w(T)$ and OPT $\geq 2 \cdot w(M)$
- Thus, cost of TSP tour returned by Christofides

  $\leq$ OPT + OPT/2

  ≤ 1.5 OPT
- Christofides is a 1.5 approximation to TSP



TSP: Summary

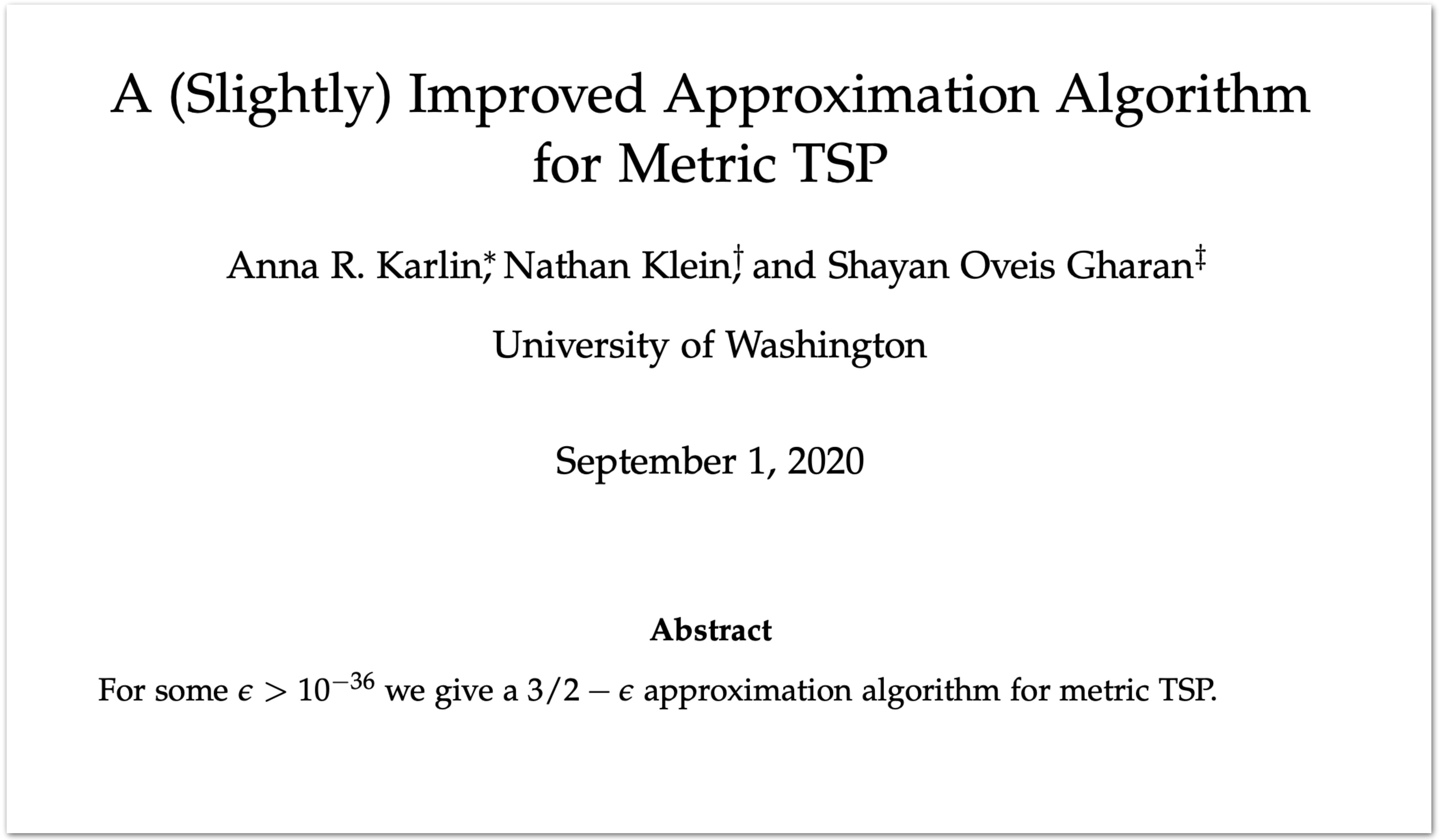
- Held & Karp [1970s] developed a heuristic for calculating a lower bound on a TSP tour (coincides with a linear program known as Held-Karp relaxation)
    - Conjectured to give a 4/3-approximation
- [Papadimitriou & Vempala, 2000's] NP-hard to approximate metric TSP within 220/219~1.0004
    - Simplified and slightly improved by Lampis'12
- "Four decades after its discovery, Christofedes' algorithm was the best approximation algorithm known for metric TSP"

No PTAS

Last Summer

- This past summer [Karlin, Klein, Shayan] (unpublished):
  - 1.4999999999999999999999999999999999 approximation
- “Euclidean TSP” does have a PTAS! [Aurora 98] [Mitchell 99]
- Understanding the approximability of TSP is a major open problem in TCS

Christofide’s isn’t optimal!



A (Slightly) Improved Approximation Algorithm for Metric TSP

Anna R. Karlin,[*] Nathan Klein,[†] and Shayan Oveis Gharan[‡]

University of Washington

September 1, 2020

Abstract

For some $\epsilon > 10^{-36}$ we give a $3/2 - \epsilon$ approximation algorithm for metric TSP.

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - Lecture slides: https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/