Approximating TSP

Admin

- “Assignment 10” (optional) review times:
    - Tuesday 9-11am
    - Wednesday 1-3pm
    - Thursday 3:30-5:30pm
    - Friday 3-5pm

- TAs have office hours if you have any general questions
    - I did give them the assignment 10 solutions but they may not be as familiar as they would be

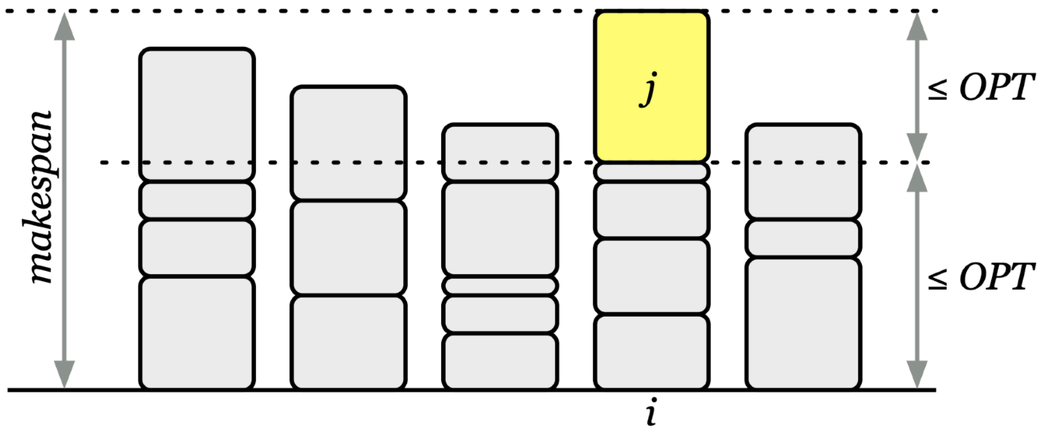
- Any questions?

Greedy is a 2-Approximation

- **Proof.**
- Consider load $L(i)$ of bottleneck machine i
- $L[i] - t_j \leq$ OPT

- We know that $t_j \leq$ OPT
- Thus, $L = L[i] \leq$ OPT$+t_j$

≤ 2OPT ∎



Greedy is a 2-Approximation

- Is our analysis tight?
- Close to it.
- Consider $m(m-1)$ jobs of length 1 and 1 job of length m
- How would greedy schedule these jobs?
  - Greedy will evenly divide the first $m(m-1)$ jobs among m machines, will place the final long job on any one machine
  - Makespan: $m - 1 + m = 2m - 1$
- How would optimal schedule it?
  - Give the long job to one machine, the rest split the other small jobs with a makespan m
- Ratio: $(2m-1)/m \approx 2$

Greedy is Online

- Notice that our greedy algorithm is an online algorithm
- Assigns jobs to machines in the order they arrive
  - Does not depend on future jobs
- Can we do better, if we assume all jobs are available at start time?
- **Offline.** Slight modification of greedy gets better approximation!

Improving on Online Greedy

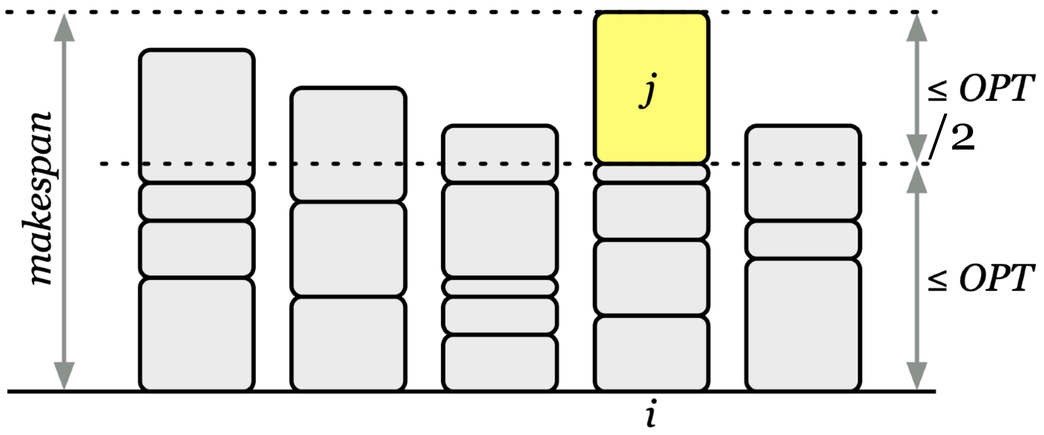
- Worst case of our greedy algorithm: spreading jobs out evenly when a giant job at the end messed things up
- What can we do to avoid this?
  - Idea: deal with larger jobs first
  - Small jobs can only hurt so much
- Turns out this improves our approximation factor
- **Longest-processing-time (LPT) first**. Sort n jobs in decreasing order of processing times; then run the greedy algorithm on them
- **Claim.** LPT has a makespan at most $1.5 \cdot$ OPT
- **Observation.** If we have fewer than m jobs, then the greedy solution is clearly optimal (as it puts each job on its own machine)

LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot$ OPT
- **Observation.**
  - If we have fewer than m jobs, then the greedy solution is clearly optimal (as it puts each job on its own machine)
- **Claim.** If more than m jobs then, OPT $\geq 2 \cdot t_{m+1}$
- **Proof.** Consider the first $m + 1$ jobs in sorted order.
  - They each take at least t_{m+1} time
  - $m + 1$ jobs and m machines, there must be a machine with at least two jobs
  - Thus the optimal makespan OPT $\geq 2 \cdot t_{m+1}$

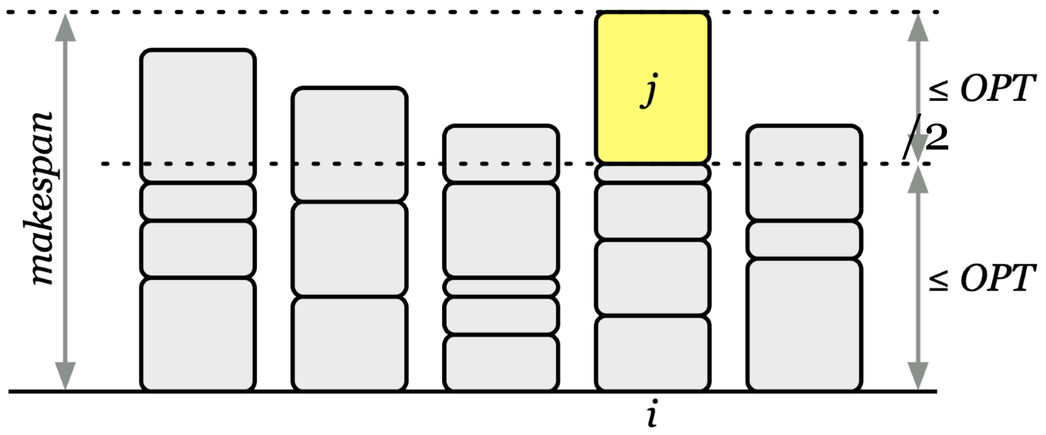
LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot$ OPT
- **Proof.** Similar to our original proof. Consider the machine M_i that has the maximum load
- If M_i has a single job, then our algorithm is optimal
- Suppose M_i has at least two jobs and let t_j be the last job assigned to the machine, note that $j \geq m + 1$ (why?)
- Thus, $t_j \leq t_{m+1} \leq \frac{1}{2}$OPT



LPT-first is a 1.5-Approximation

- **Lemma.** LPT-first has a makespan at most $1.5 \cdot$ OPT
- **Proof.** Similar to our original proof. Consider the machine M_i that has the maximum load
- If M_i has a single job, then our algorithm is optimal
- Suppose M_i has at least two jobs and let t_j be the last job assigned to the machine, note that $j \geq m + 1$ (why?)
- Thus, $t_j \leq t_{m+1} \leq \frac{1}{2}$OPT
- $L[i] - t_j \leq$ OPT
- $L[i] \leq \frac{3}{2}$OPT ∎



Is our 1.5-Approximation tight?

- **Question.** Is out $3/2$-approximation analysis tight?
  - Turns out, no
- **Theorem [Graham 1969].** LPT-first is a $4/3$-approximation.
  - Proof via a more sophisticated analysis of the same algorithm
- **Question.** Is the $4/3$-approximation analysis tight?
  - Pretty much.
- Example
  - m machines, $n = 2m + 1$ jobs
  - 2 jobs each of length $m, m+1, \ldots, 2m-1$ + one job of length m
  - Approximation ratio $= (4m-1)/3m \approx 4/3$

Can we do better than $4/3$?

- Long series of improvements
- Polynomial time algorithm for *any* constant approximation [Hochbaum Shmoys 87]
- Specifically: $(1+\epsilon)$ approximation in $O\left((n/\epsilon)^{1/\epsilon^2}\right)$ time
- **PTAS:** Polynomial time approximation scheme
- For any desired constant-factor approximation, there exists a polynomial-time algorithm

Approximate TSP

Approximating TSP

- Recall the traveling salesman problem
- n cities labeled $v_1, \ldots, v_n$
- Let $d(i, j)$ be the distance from city v_i to city v_j
- **TSP. (Decision Version)** Given pairwise distance between n cities and a bound D, is there a tour (that visits each city exactly once and returns to starting city) of length at most D?
- **NP complete problem.** 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 $(v_i, v_j) \notin E$

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 $c \cdot \mathrm{OPT}$
- 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 $(v_i, v_j) \in E$
  - $d(c_i, c_j) = cn + 1$ 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 where the distances satisfy the triangle inequality, that is,
  - $d(i, j) \leq d(i, k) + d(k, j)$ for any cities i, j, k
- Other properties of Euclidean (metric) distances:
  - $d(i, i) = 0$ and $d(i, j) \geq 0$ [Identify and Non-negative]
  - $d(i, j) = d(j, i)$ [Symmetric]
- Note that **Metric TSP** is still NP complete (reduction from undirected hamiltonian cycle)
  - Setting $d(c_i, c_j) = 2$ when $(v_i, v_j) \notin E$ satisfies triangle inequality

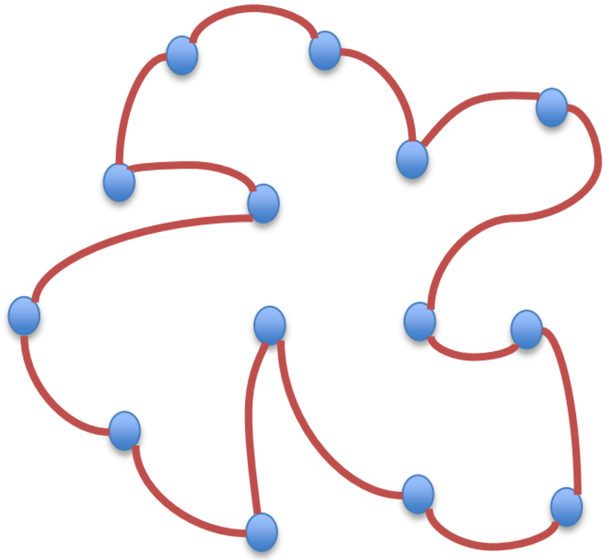
Approximating Metric TSP

- 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
- To approximate, consider the optimization version of the problem
- **Goal.** Find the tour of min total distance that visits each city once
- Steps to follow when designing an approximation algorithm for a minimization problem (NP hard)?
    - **Lower bound the optimal cost** by some function of input
    - Upper bound the cost of algorithm by **the same function**
- Minimum spanning trees give us such upper/lower bounds
- We give a 2-approximation to metric-TSP using minimum spanning trees

Lower Bound on OPT

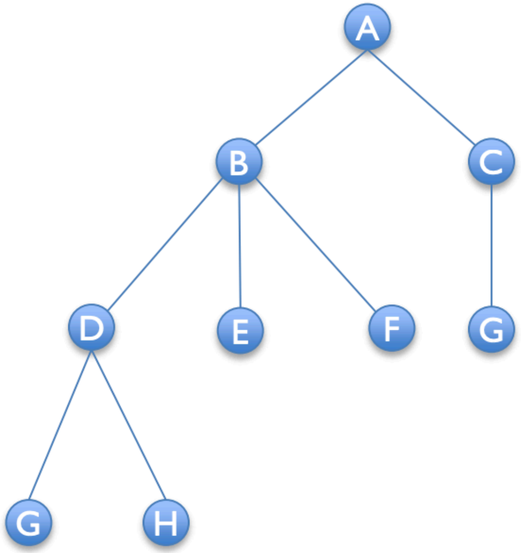
- **Claim**. Let T be the minimum spanning tree of G then length of the optimal tour OPT $\geq w(T)$.

- **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 ∎



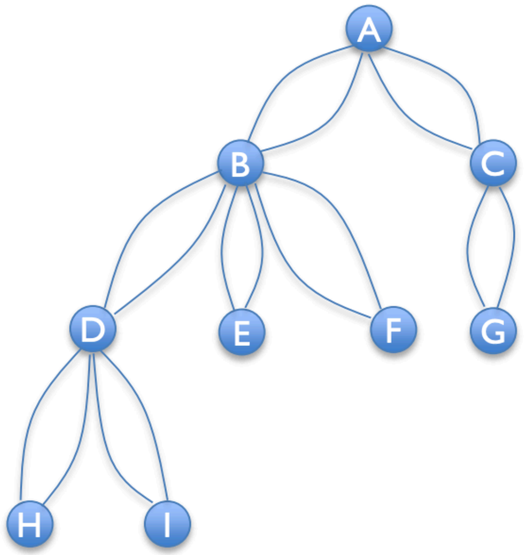
Double Tree Algorithm

- **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

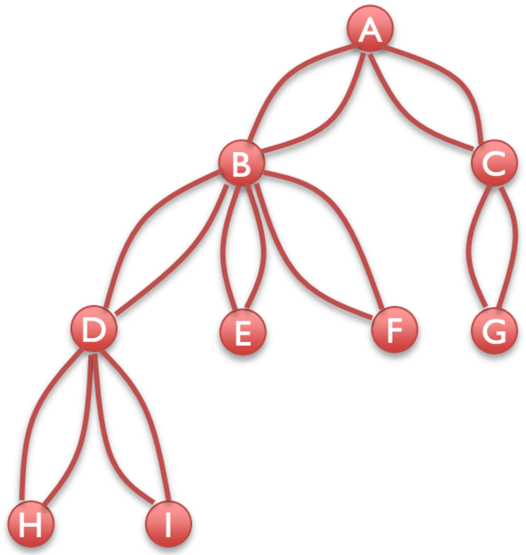
- Find a minimum spanning tree T
- **Duplicate every edge in T**
- Find an Eulerian tour of resulting multi-graph
- Shortcut Euler tour to avoid repeated vertices



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

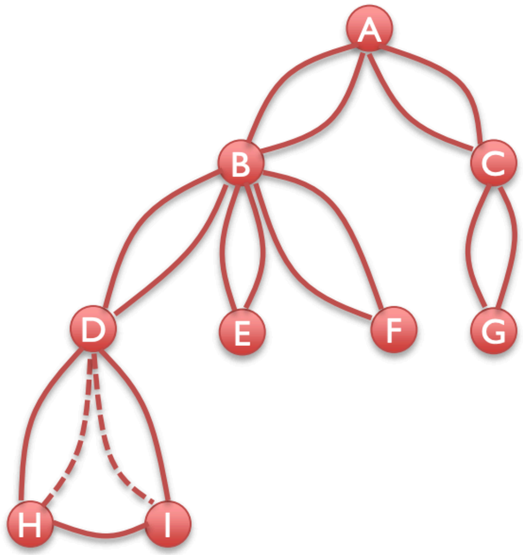
Why must an Euler tour exist?



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

Double Tree Algorithm

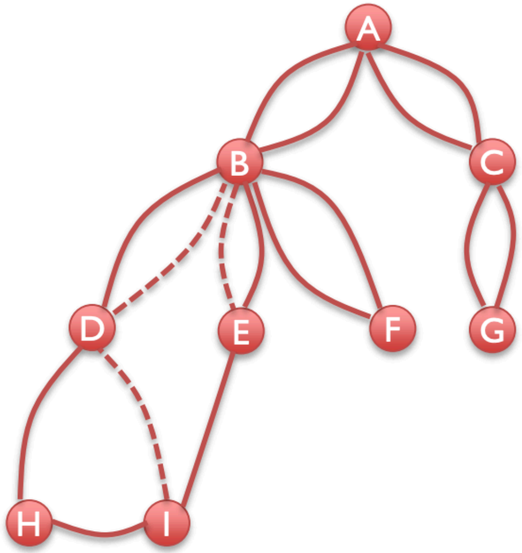
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



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

Double Tree Algorithm

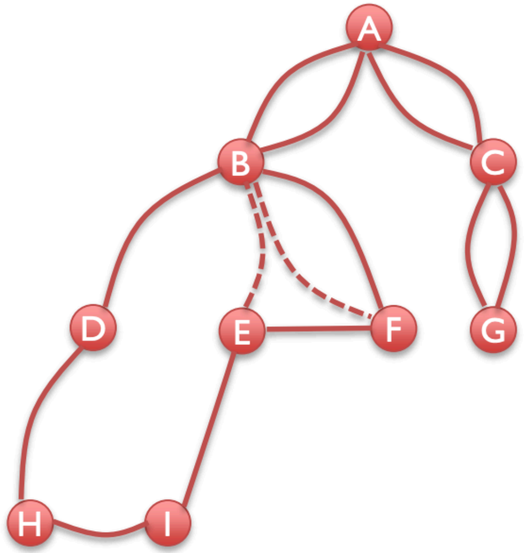
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



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

Double Tree Algorithm

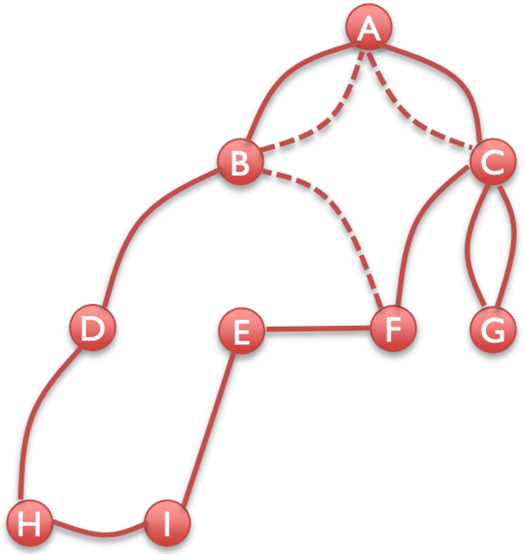
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



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

Double Tree Algorithm

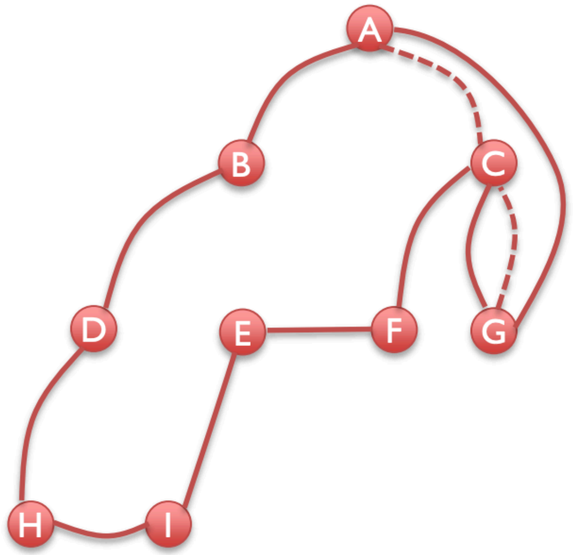
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



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

Double Tree Algorithm

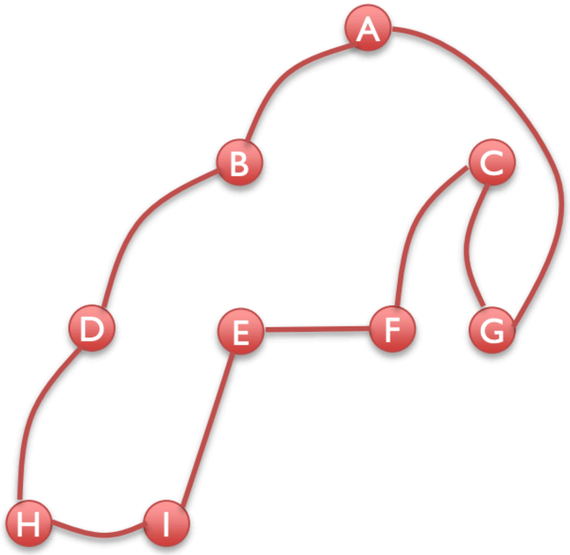
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



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

Double Tree Algorithm

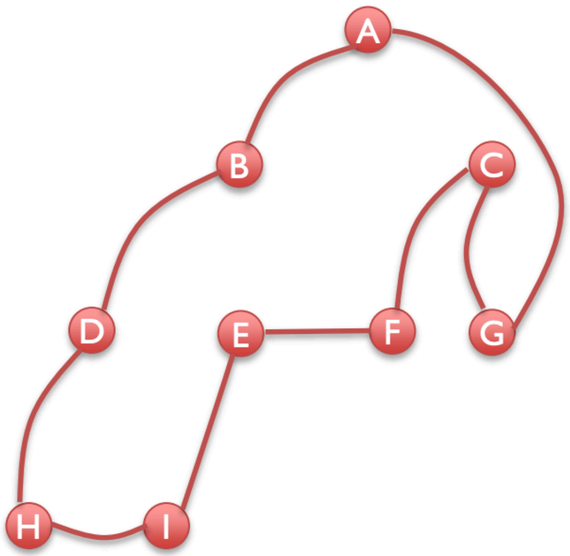
- Find a minimum spanning tree T
- Duplicate every edge in T
- Find an Eulerian tour of resulting multi-graph
- **Shortcut Euler tour to avoid repeated vertices**



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

Double Tree Analysis

- **Claim.** The 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 $\leq 2 \cdot w(T)$
- Due to triangle inequality, shortcutting the tour does not increase length
- Since $w(T) \leq$ OPT, we get that our tour length is $\leq 2 \cdot$ OPT
  ∎



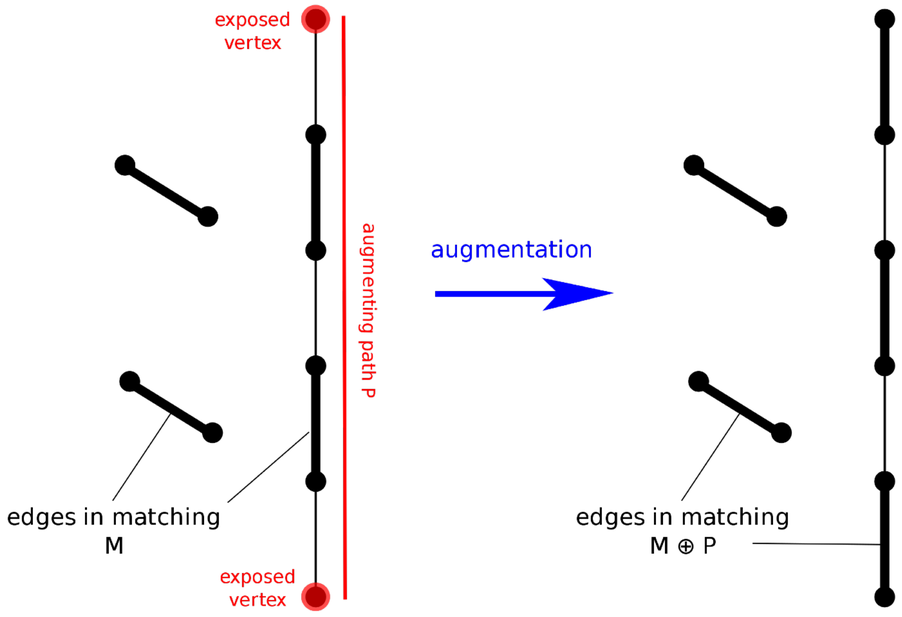
A,B,D,H,~~D~~,I,~~D~~,~~B~~,E,~~B~~,F,~~B~~,~~A~~,C,G,~~C~~,A

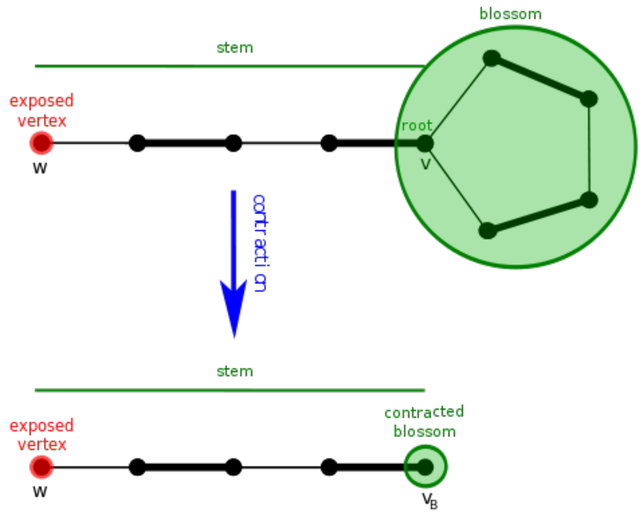
Christofides Algorithm [Christofides 76][Serdyukov 76]

- Doubling the edges of MST is one way to obtain Euler tour of the MST, but is there a cheaper way to augment to tree to obtain an Eulerian tour?
- A graph has an Euler tour iff all nodes have even degree
- 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



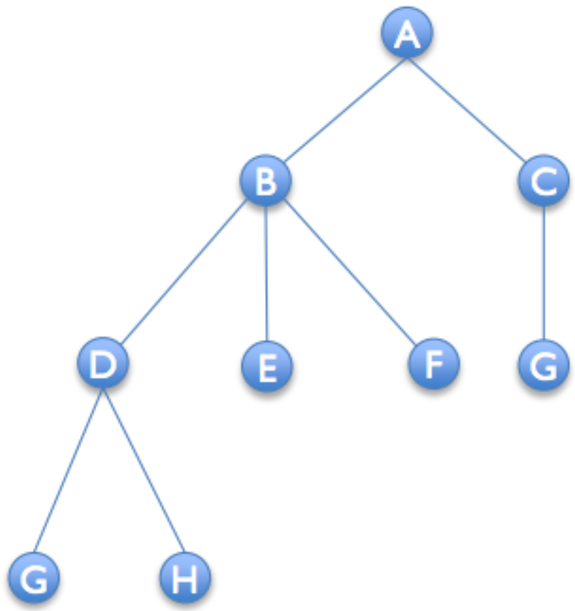


Christofides Algorithm

- 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

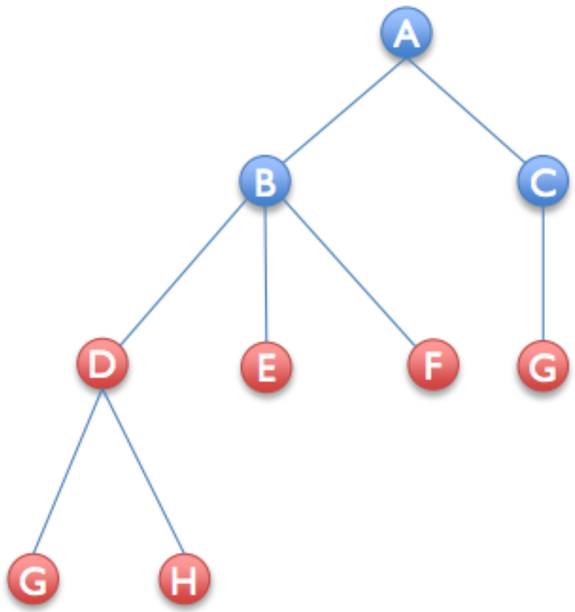
All odd-degree vertices and any edges connecting them



Christofides Algorithm

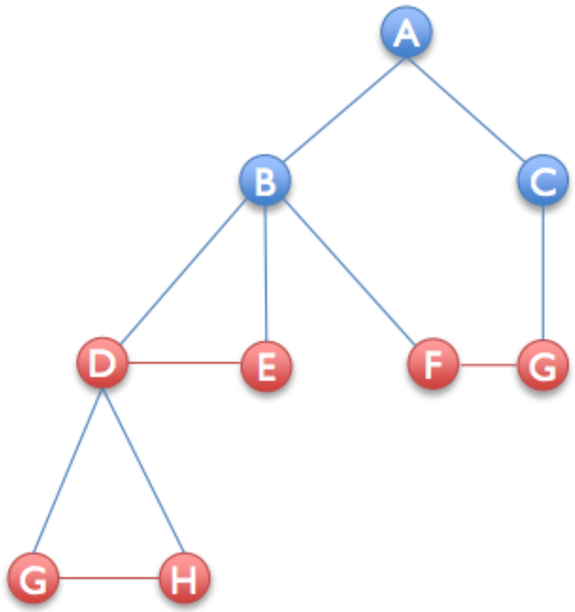
- Find the minimum spanning tree T
- Compute O: the set of odd degree vertices in T
- Find the min-cost perfect matching M of subgraph induced by O
- Return shortcut of Euler tour of $T \cup M$

What does adding M do to O?



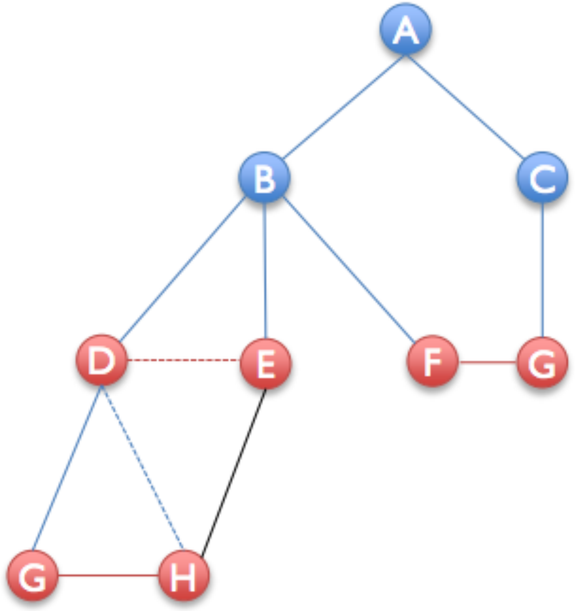
Christofides Algorithm

- 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 $T \cup M$



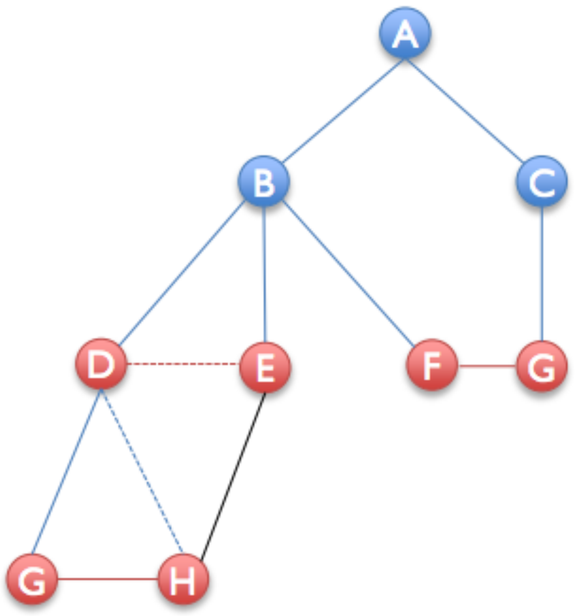
Christofides Algorithm

- 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 $T \cup M$



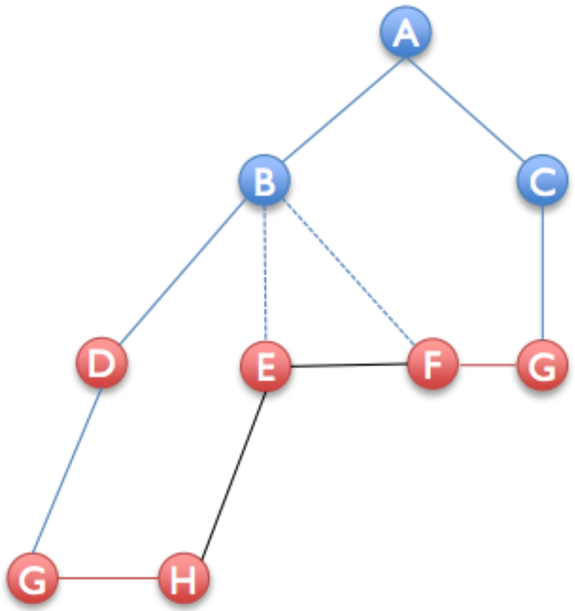
Christofides Algorithm

- 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 $T \cup M$



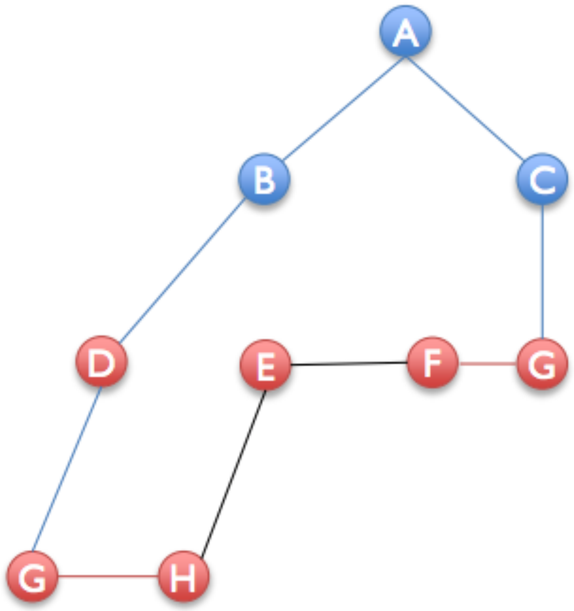
Christofides Algorithm

- 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

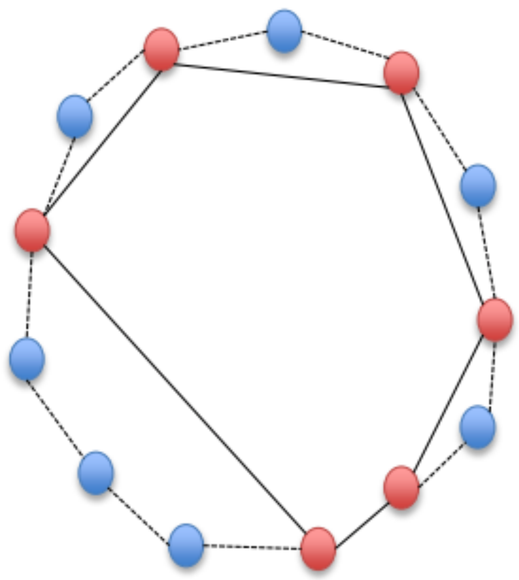
- Cost of TSP tour returned is at most $w(T) + w(M)$
- We know OPT $\geq w(T)$
- To bound the approximation factor, we lower bound the OPT in terms of the cost of M
- **Claim.** Let OPT be the length of the optimal tour and let M be a minimum-cost perfect matching on the complete subgraph induced by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

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

- Once we prove the lemma, we have, $w(T) + w(M) \leq \frac{3}{2} \cdot \text{OPT}$
- Thus, Christofides algorithm is a $3/2$-approximation to metric TSP

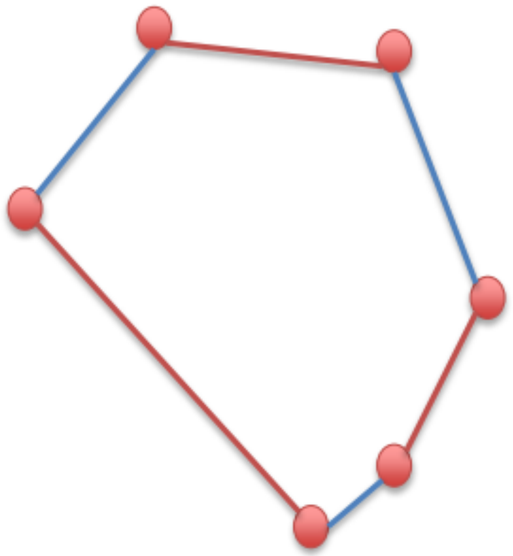
Christofides Analysis

- **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$/2$



TSP: Summary

- Held & Karp [1970s] developed a hueristic 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 is the best approximation algorithm known for metric TSP"
- This past summer [Karlin, Klein, Shayan] (unpublished):
    - 1.499999999999999999999999999999999999 approximation
- "Euclidean TSP" does have a PTAS! [Aurora 98] [Mitchell 99]
- Understanding the approximability of TSP is a major open problem in TCS

No PTAS

Christofide's isn't optimal!

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/