Facility Location

The location of a set of facilities should be determined. These facilities serve clients and we want them to be as close as possible to the clients.

facilities can be…

- factories, warehouse, retailers, servers.

  objective: min sum (or average) of distances.

- hospitals, police stations, fire-stations, antennas

  objective: min maximal distance.

Facility Location

Various questions:

- Where should a facility be?
- How many facilities should we build?
- How should demand be allocated?

Problems get more complex (adding constraints)

- warehouse capacities
- each client can be allocated to only one warehouse
- different costs (transportation, holding, operating, set-up)
- distance / service time
- dynamic clients.

Different types of Location Models

planar

- facilities can be located anywhere on a plane
- generally continuous optimization problems

network

- Facilities and travel are restricted to some underlying network (ex. roads, cables).
- Facilities may be restricted to nodes in the network, or may be also along edges.

discrete

- facilities can only be at a set of potential locations.

Algorithms for FL Problems

We will see that some FL problems are easy to solve while others are NP-hard, and approximate solutions will be used.

Optimal solutions are not always necessary.

- may take too much effort with very little effect on the result.
- often input data is estimated anyway.

The Planar Model

How do we measure distance?

- Manhattan or Rectilinear norm

  $d_{ij} = |x_i - x_j| + |y_i - y_j|$

  Useful for cities that are set up in grids or factory floors with aisles.

- Euclidean

  $d_{ij} = \mathrm{sqrt}((x_i - x_j)^2 + (y_i - y_j)^2)$

  Always gives a lower bound on the actual distance.

- l_p norm

  $d_{ij} = ((x_i - x_j)^p + (y_i - y_j)^p)^{1/p}$

  Generalizes previous measures.

Possible Planar Models

Given existing clients and facilities (maybe none), we want to introduce new facilities.

1. Single Facility minisum: Locate a single new facility to minimize sum of distances between clients to new and existing facilities.
2. Single Facility minimax: Locate a single new facility to minimize max distance between a client and new or already existing facilities.
3. Multifacility minisum: same as 1, but with multiple new facilities.
4. Multifacility minimax: same as 2, but with multiple new facilities.

Planar Single Facility Rectilinear Distance MiniSum Location Model

- m existing customers $\{(a_i, b_i)\}$
- customer i is serviced w_i times per month
- locate a single service facility to minimize total distance traveled.

Solution: Assume the facility is located at (x,y).

$$f(x,y) = \sum_{i=1}^{m} w_i \, d((x,y),(a_i,b_i))$$
$$= \sum_{i=1}^{m} w_i \, (|x-a_i|+|y-b_i|).$$

Let $f_1(x)= \sum_{i=1}^{m} w_i \, (|x-a_i|)$; $f_2(y)= \sum_{i=1}^{m} w_i \, (|y-b_i|)$.

Note: $f(x,y) = f_1(x)+f_2(y)$ so we can consider x and y dimensions independently.

Planar Single Facility Rectilinear Distance MiniSum Location Model

Example: (a,b) = {(7,10), (15,7), (15,3), (12,6), (0,5)}

w = { 160, 40, 60, 140, 180}

$f_1(x) = 160 \cdot |x-7| + 100 \cdot |x-15| + 140 \cdot |x-12| + 180 \cdot |x-0|$

Plot to find optimal value of x.

For the range $7 \leq x \leq 12$:

$f_1(x) = 160(x-7)+100(15-x)+140(12-x)+180x =$

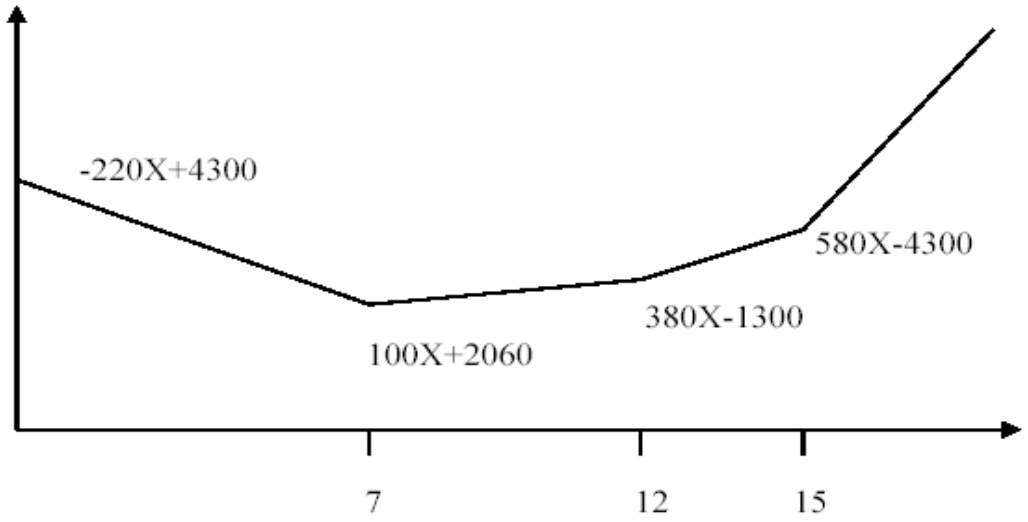
$100x+2060$

Similarly, find $f_1(x)$ for all x ranges.

Planar Single Facility Rectilinear Distance MiniSum Location Model

$f_1(x) = 160 \cdot |x-7| + 100 \cdot |x-15| + 140 \cdot |x-12| + 180 \cdot |x-0|$

Plot:



x=7 is optimal.

Use the same method to find optimal y.

(7,6) is the optimal location.

Graphs or Networks

Context: a set of points, represented by vertices, and a set of links, represented by edges, connecting pairs of those points.

Applications: communication network, highway system, social networks.

Definitions:

1. *A path* between two nodes is a sequence of edges leading from one node to another.
2. *A cycle* is a path that leads back to starting node.
3. *A tree*: a connected graph with no cycles.

Network/Discrete models

1. Covering: how many facilities should we build so that each customer is within a given distance of its nearest facility?

   Example: fire stations.

2. Center Models (k-center problem)

   Minimize the max distance between facilities and customers (between a customer and its nearest facility).

3. Median Models: (k-median model)

   Minimize the sum of distances between customers and their nearest facility.

   Example: warehouse problem

Network/Discrete models

4. Integrated Production-Distribution Models
   - Potential facilities have costs.
   - Minimize total cost of facilities and shipping.

5. Routing Models

   A set of (possibly capacitated) vehicles should travel from a facility to a set of customers to make a delivery and back to the facility. Route the vehicles so that all demand is met and total distance is minimized.

Example: Traveling Salesman Problem

Covering a Network

Covering: how many facilities should we build so that each customer is within a given distance from its nearest facility?

Possible problems:

- Each client has its own requirement, or all clients have the same requirement.
- Facilities can be located only on vertices or any point in the network.

Covering a Network

Theorem: The network covering problem is NP-hard.

Proof: Reduction from Dominating Set.

A dominating set in an undirected graph is a collection S of vertices with the property that every vertex v in G is either in S, or there is an edge between a vertex in S and v.

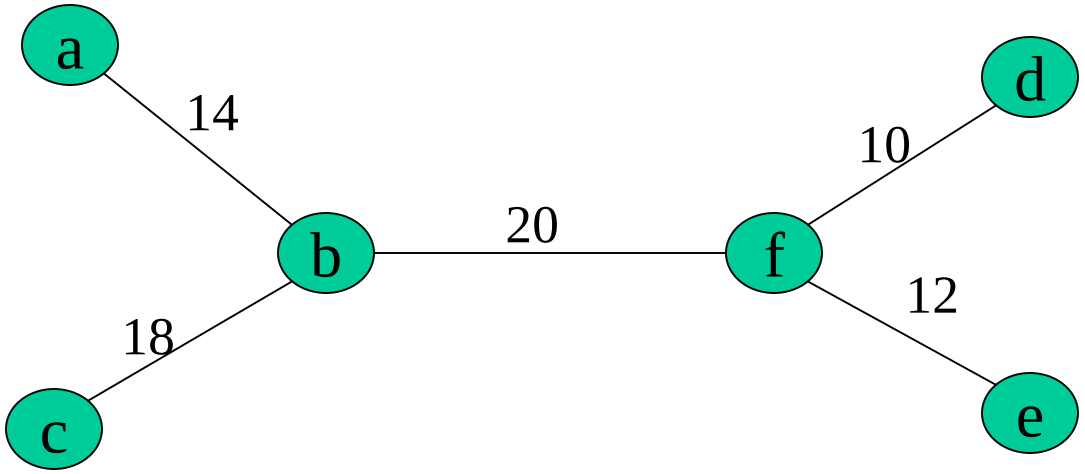
Given a graph G and an integer k, it is NP-hard to determine whether G has a DS of size k.

We show that DS $\leq_p$ Covering :

Given an instance of DS we build an instance of the covering problem: Same network, same k, all edges have length 1.

Covering a tree using a minimal number of facilities

When the network is a tree (i.e. there are no cycles), there exists a simple algorithm to find an optimal solution to the covering problem.



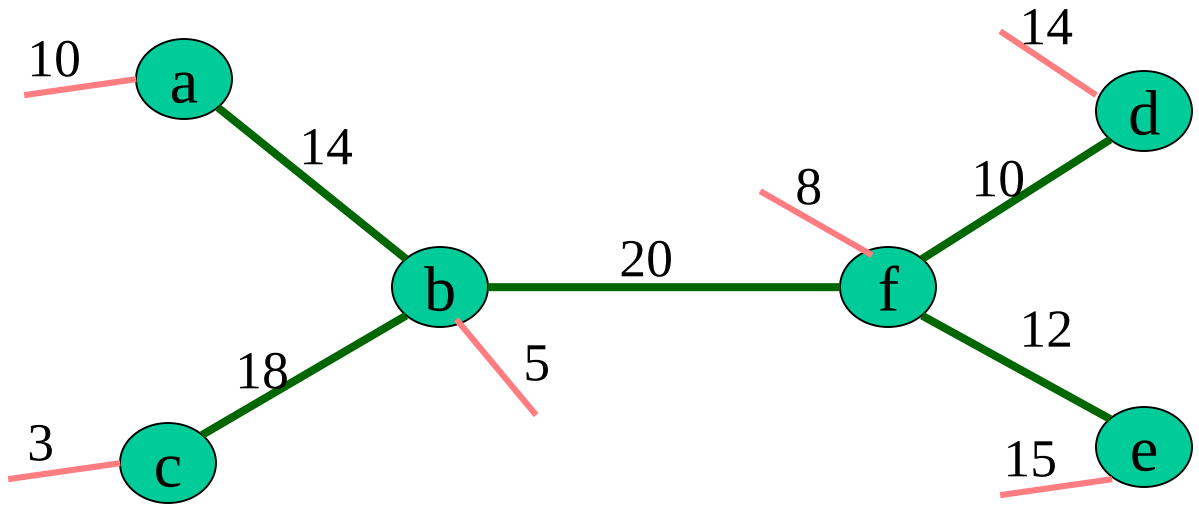
Input: A weighted tree, each vertex i needs to be within some distance s_i from a center.

s_a=10 ; s_b=5 ; s_c=3 ; s_d=14 ; s_e=15 ; s_f=8

Covering a tree.

Output: location of centers.

Goal: A cover with minimal number of centers.



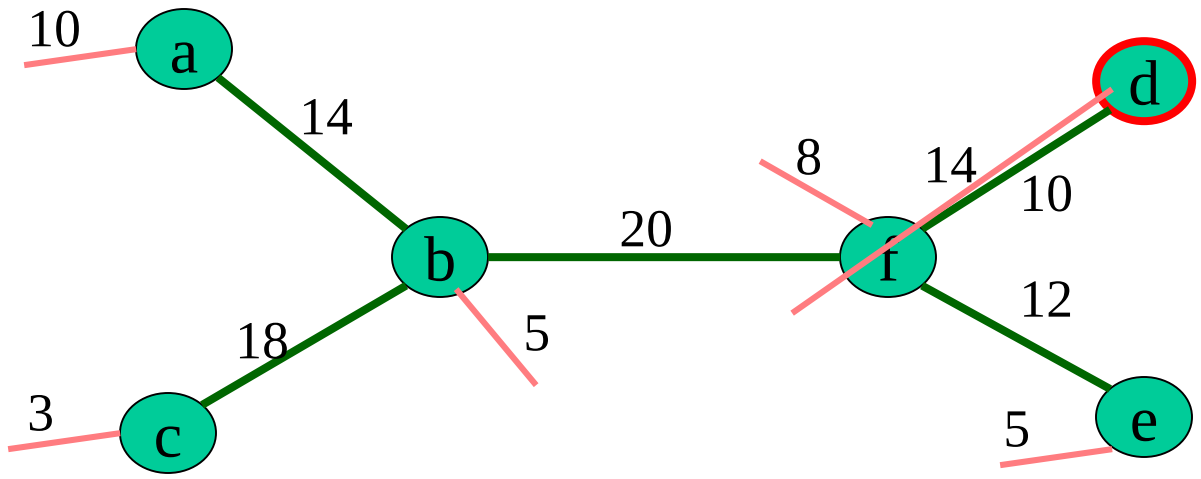
s_a=10
s_b=5
s_c=3
s_d=14
s_e=15
s_f=8

Step 1: attach a "string" of length s_i to vertex i.

Mark all the tree as non-processed (green).

Covering a tree.

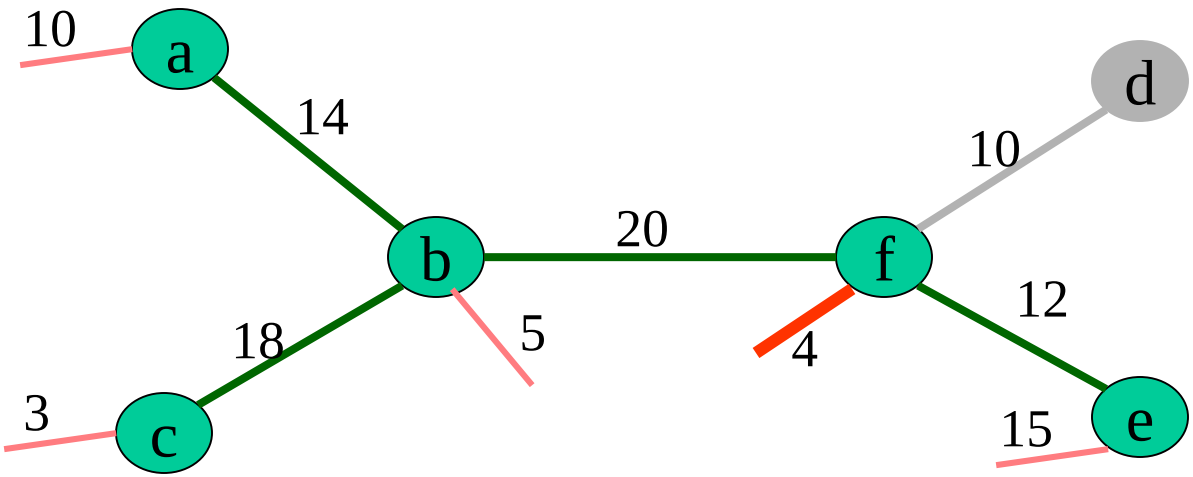
Step 2: pick an arbitrary leaf v, check if it is already covered by existing centers. If not, 'stretch' its string towards its neighboring vertex u. If it reaches u, s_u = min (s_u, excess). If it doesn't reach u, add a facility.



s_a=10
s_b=5
s_c=3
s_d=14
s_e=15
s_f=8

Example: select d for active leaf. Stretch the string towards f. Excess=4, update s_f =14-10=4.

Covering a tree.

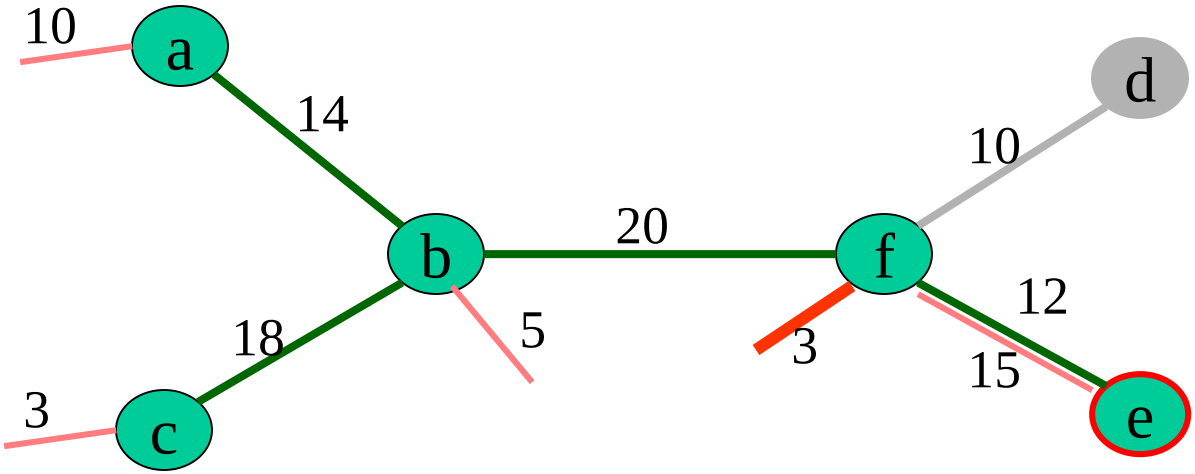


Step 3: remove v and the edge (u,v) from the graph (color them gray).

If the graph is not empty, go to step 2.

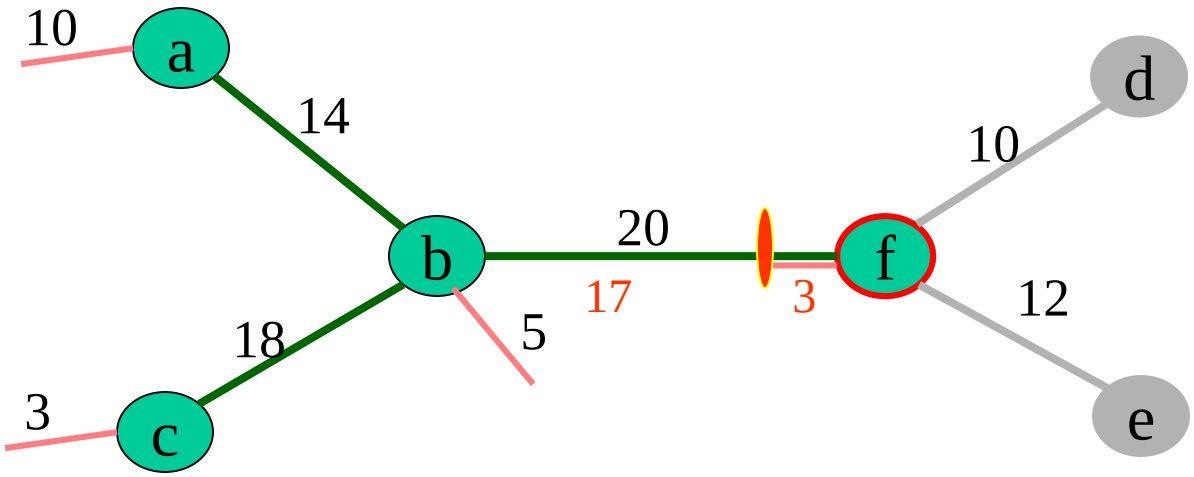
Covering a tree.

v=e, s_e=15, Excess=3



s_f is reduced from 4 to 3

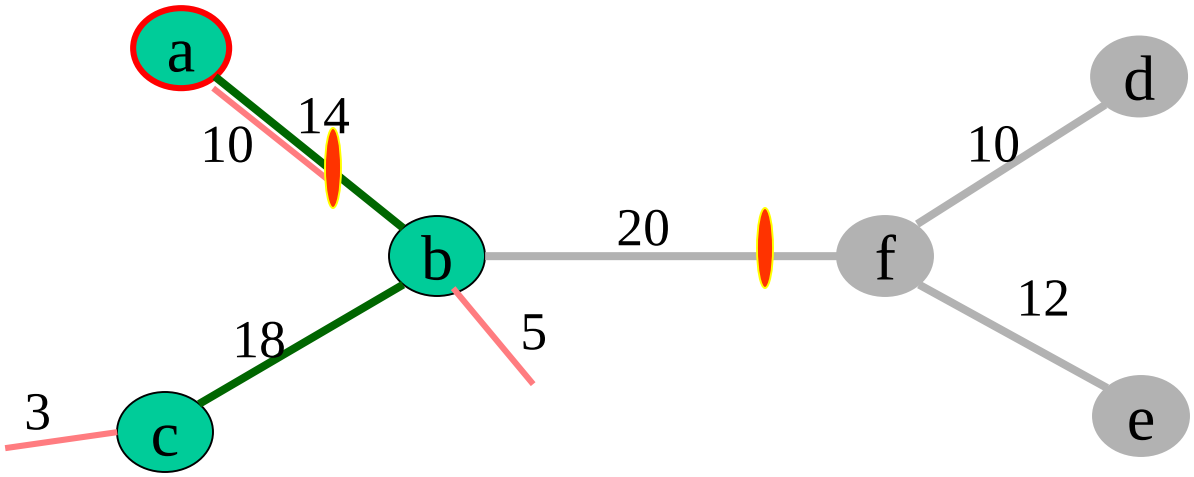
Covering a tree.



v=f. s_f=3, No Excess.

Place a center along f-b. 3 units from f

Covering a tree.

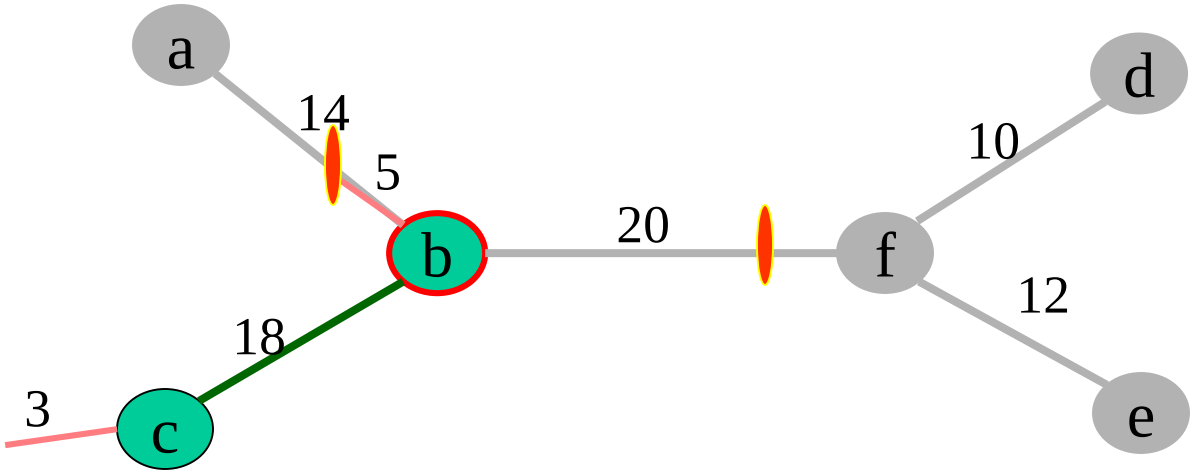


v=a. s_a=10.

Check if a is already covered by any center (no)

No Excess. Place a center along a-b.

Covering a tree.

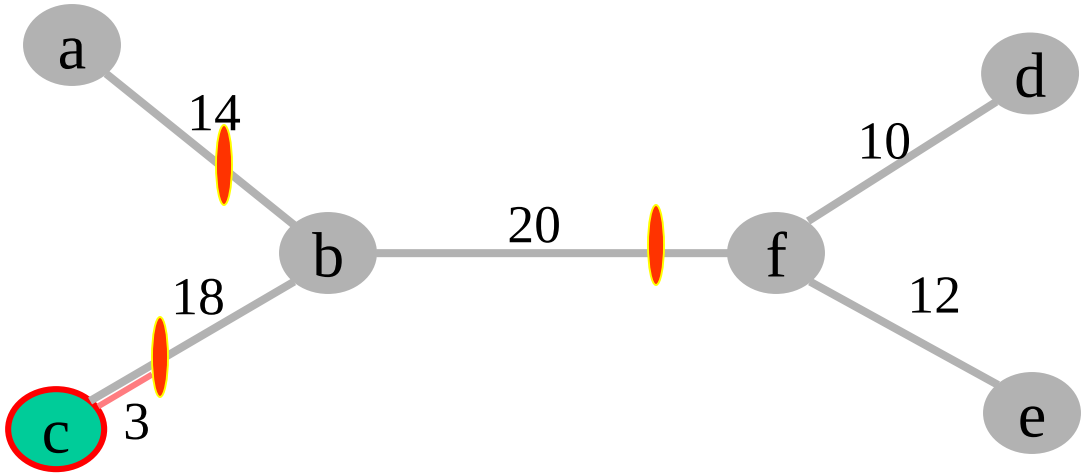


v=b. s_b=5.

Check if b is already covered by any center (yes!)

Remove b from the graph.

Covering a tree.

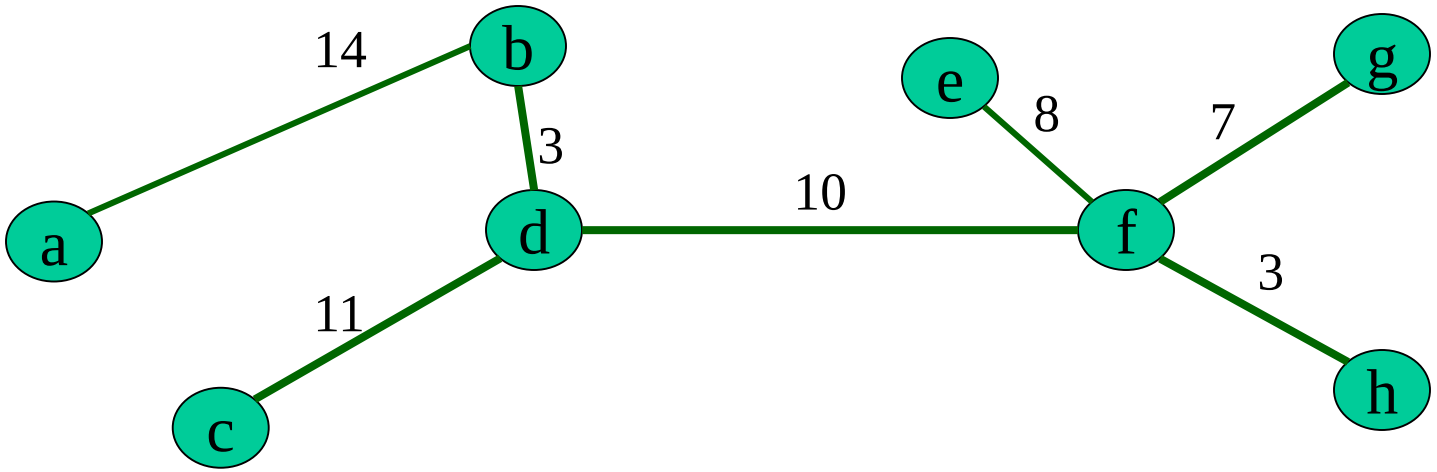


v=c. s_c=3, No active neighbor
Check if c is already covered by any center (no)

can place a center anywhere along (c-b) within distance 3 from c

The whole graph is covered (gray) using 3 centers.

In class exercise: find an optimal covering.



$s_a=18,\ s_b=5,\ s_c=10,\ s_d=2,$
$s_e=5,\ s_f=4,\ s_g=10,\ s_h=6$

Covering a tree.

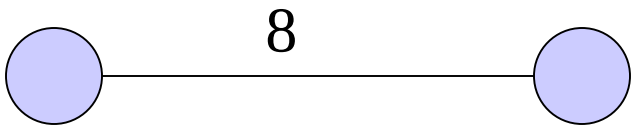
Theorem: The algorithm is optimal, that is, it uses the minimal possible number of centers.

Proof: In class.

k-center Problems

Objective: Locate k facilities (centers) so as to minimize the maximum distance between a customer and its nearest facility (a Minimax Problem).

For problems on networks, we distinguish between Vertex Center Problems and Absolute (or General) Center Problems.



Vertex Center Problem: opt =8

Absolute Center Problems: opt =4

1-Center in a general network,

Let x be any point on the network. The distance between x and the node of G which is farthest away from it is denoted $m(x) = \max_{v \in V} d(x,v)$.

Definition: A point x_e on an edge (p,q) is a local center if for every $x \in (p,q)$ including the nodes p,q

$$m(x_e) \leq m(x)$$

Definition: A vertex $v^* \in V$ is a vertex center of a network G, if for every $v \in V$, $m(v^*) \leq m(v)$.

Definition: A point $x^* \in G$ is an absolute center of a network G, if for every point $x \in G$, $m(x^*) \leq m(x)$.

Finding a vertex 1-Center in a network

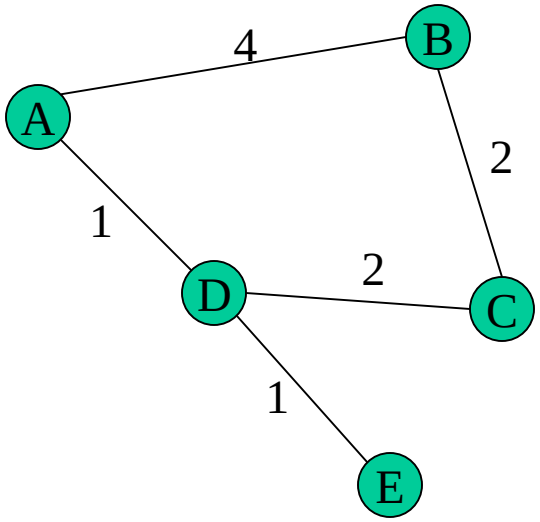
An optimal Algorithm:

1. Compute the matrix of shortest paths costs for all pairs of nodes.

2. The vertex 1-center is the one with the smallest maximum value in its row in the matrix.

Vertex 1-Center in a network, Example

Input:



Shortest path matrix:

$$[d(i,j)] = \begin{array}{c} \text{From} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \text{To} \\ \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 4 & 3 & 1 & 2 \\ 4 & 0 & 2 & 4 & 5 \\ 3 & 2 & 0 & 2 & 3 \\ 1 & 4 & 2 & 0 & 1 \\ 2 & 5 & 3 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{l} \text{Maximum in Row} \\ 4 \\ 5 \\ 3 \leftarrow \text{minimum} \\ 4 \\ 5 \end{array}$$

C in the optimal center.

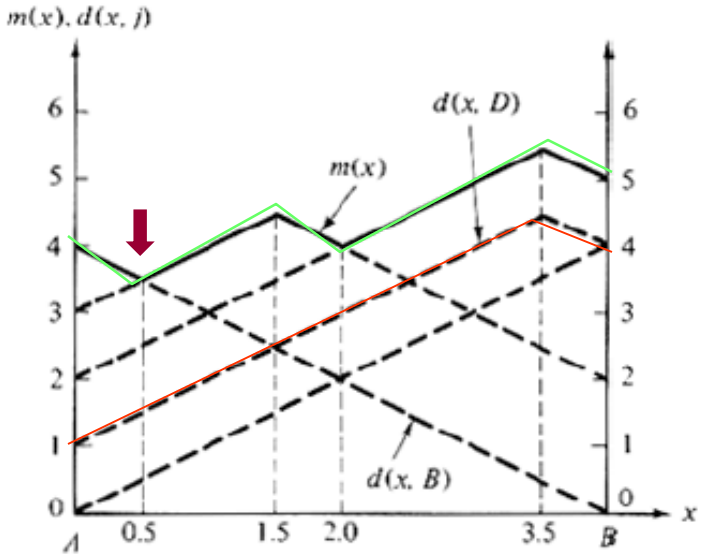
Absolute 1-Center in a network.

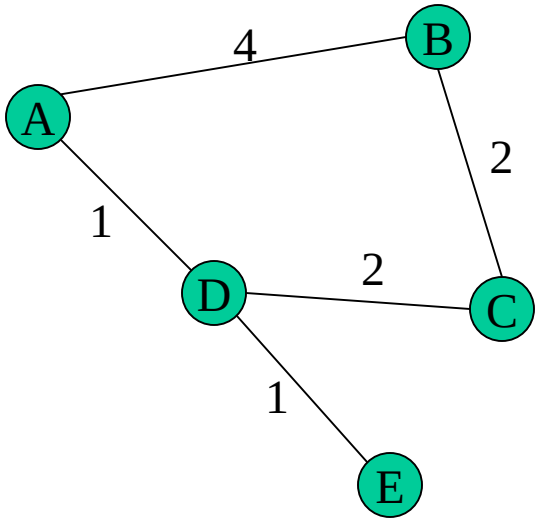
STEP 1: For each edge e of G. find the local center x_e of e.

STEP 2: Among all the local centers x_e , choose the one with the smallest $m(x_e)$ value. That local center is also the absolute center x^* of G.

Step 1 is time consuming, as we will demonstrate for our example network.

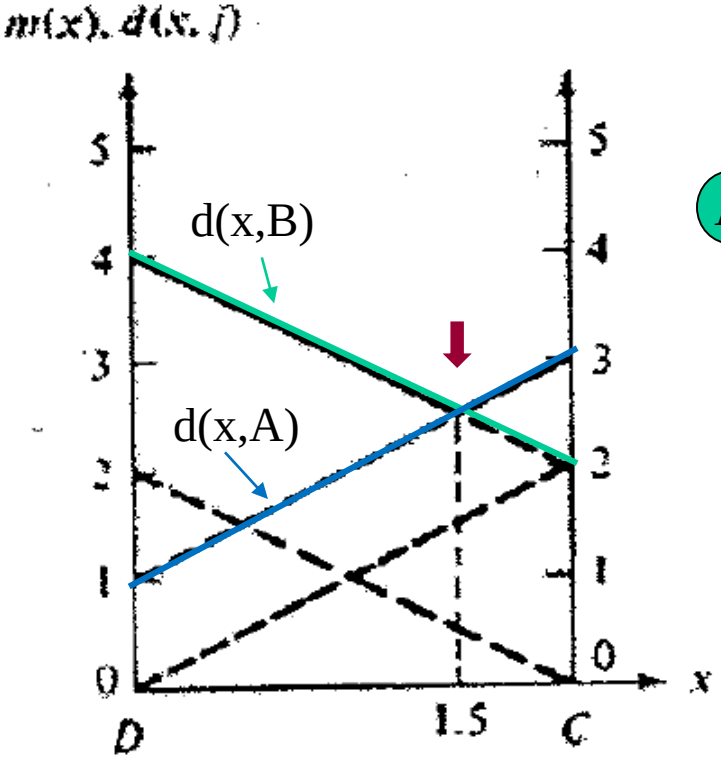
Finding a local center of an edge.



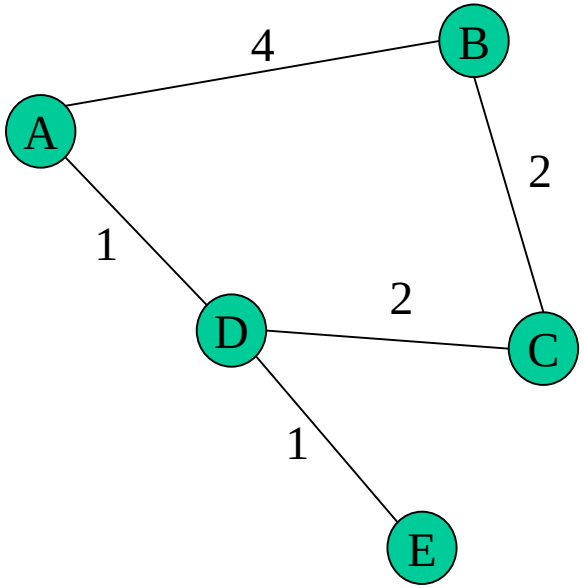


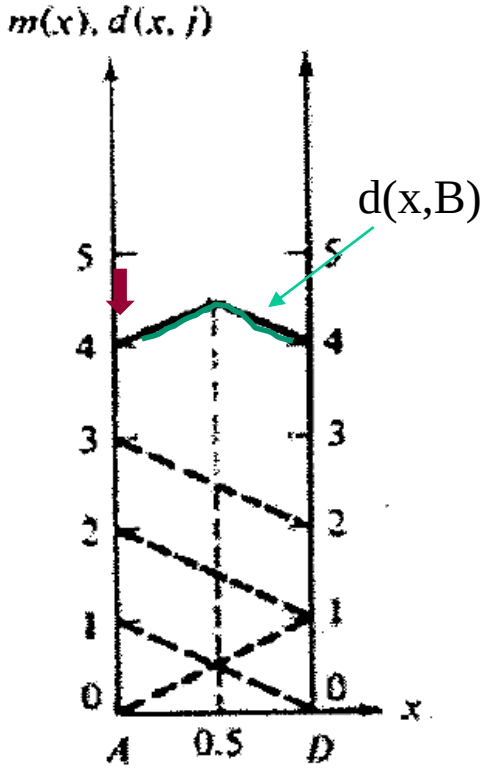
For e=(A, B), For all points x on (A, B), find and plot the functions d(x,j) for j = A,.., E. By convention, set x= 0 at A and x = 4 at B. Since m(x) = Max (d(x,A),…, d(x, E)), the function m(x) is given by the upper envelope of the five functions. The local center of this edge is at a point 0.5 unit away from A (and 3.5 units away from B) and m(x) = 3.5.

Finding a local center of an edge.



Graph for (D,C)





Graph for (A,D)

Absolute 1-Center in a network.

(A,B): local center 0.5 unit from A; m(x) = 3.5

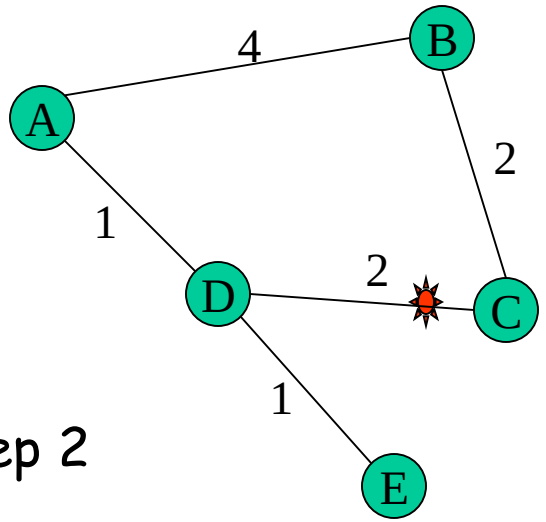
(A,D): local center at A and at D; m(x) = 4

(B.C): local center at C; m(x) = 3

(D,E): local center at D; m(x) = 4

(C,D): local center 0.5 unit from C; m(x) = 2.5

This completes Step 1 of the algorithm. In Step 2 we choose the x* on the link (C, D) m(x*) = 2.5.



Remark: The absolute center and the vertex center do not have to coincide. In fact, the absolute center *does not* have to be on a link adjacent to the node where the vertex center is located.

Absolute 1-Center in a network.

In order to reduce the complexity of examining all local centers of the network edges, we use the following theorem:

Theorem: For the local center, x_e , on an edge (p,q),

$$m(x_e) \geq \frac{m(p) + m(q) - c(p,q)}{2}$$

where, c(p, q) denotes the length of (p,q).

Proof: In class.

How can this theorem be used?

Improved Algorithm for Absolute 1-Center.

STEP 1: find a vertex center $x \in V$, let $t=m(x)$

STEP 2: For each edge (p,q) in sequence do

If $t > \frac{m(p)+m(q)-c(p,q)}{2}$ then find the local center x_{pq} of e and let $t = \min\{t, m(x_{pq})\}$.

STEP 3: Output t and x^*, where x^* is the last point that assigned a value to t, and $t=m(x^*)$.

In the example, the local center need be found only for (A,B) and (C,D).

k-center Problem

Input:

- An undirected network G with n nodes.
- A positive integer k.

Output: k distinct points on the graph, to be denoted $X_k = \{x_1, x_2, \ldots, x_{k-1}, x_k\}$.

Goal: Let $d(X_k,v)$ be the minimum distance between any one of the points $x_1 \ldots x_k$ and the node v on G. The goal is to minimize

$$\max_{v \in V} d(X_k, v)$$

k-center Problem

Theorem: The k-center problem is NP-hard.

Proof sketch: A reduction from *dominating set*.

A dominating set in an undirected graph is a collection S of vertices with the property that every vertex v in G is either in S, or there is an edge between a vertex in S and v.

Observation: A dominating set of size k is a k-center with objective value =1.

The Metric k-center Problem

G is undirected and obeys the triangle inequality

For all $u,v,w \in V$: $d(u,w) \leq d(u,v)+d(v,w)$.

A 2-approximation greedy algorithm for this problem:

1. Choose the first center arbitrarily.
2. At every step, choose the vertex that is furthest from the current centers to become a center.
3. Continue until k centers are chosen.

Example: In class.

The Metric k-center Problem

Analysis of 2-approximation algorithm:

- Note that the sequence of distances from a new chosen center, to the closest center to it (among previously chosen centers) is non-increasing.
- Consider the point that is furthest from the k chosen centers.
- We need to show that the distance from this point to the closest center is at most $2 \cdot OPT$.
- Assume by contradiction that it is $> 2 \cdot OPT$.
- This means that distances between all centers are also $> 2 \cdot OPT$.

The Metric k-center Problem

Analysis of 2-approximation algorithm (cont'):

- We have k+1 points with distances > 2 ·OPT between every pair.
- Each point has a center of the optimal solution with distance ≤ OPT to it.
- There exists a pair of points with the same center X in the optimal solution (pigeonhole principle: k optimal centers, k+1 points)
- The distance between them is at most 2·OPT (triangle inequality).
- Contradiction!

The Metric k-center Problem

Theorem: If P≠NP, then there is no approximation algorithm for the metric k-center that gives a (2-ε)-approximation for ε>0.

Proof: Reduction from dominating set. We will see: a (2-ε)-approximation to the metric k-center problem can be used to solve dominating set optimally.
Let $G = (V,E)$, k be an instance of the dominating set problem.
Define a complete graph $G' = (V,E'=V\times V)$, where
$d(u,v)=1$, if $(u,v) \in E$
$d(u,v)=2$, if $(u,v) \notin E$

Note: G' satisfies the triangle inequality.

The Metric k-center Problem

Run the (2-ε)-approximation algorithm for metric k-center problem on G'. Let d be the objective value of the selected set.
If $d \leq 1$ then return 'TRUE' (G has a DS of size k).
If $d \geq 2$ then return 'FALSE'.

Explanation: Suppose G has a dominating set of size at most k. Then G' has a k-center with objective value at most 1.
A (2-ε)-approximation algorithm must select a set with $d < 2$.
If there is no DS of size k in G, every k-center has objective value $\geq 2 > 2-\varepsilon$.

k-median Problem

Input:

- An undirected network G with n nodes, each node has a weight h_j.
- A positive integer k.

Output: k distinct points on the graph, to be denoted $X_k = \{x_1, x_2, \ldots, x_{k-1}, x_k\}$.

Goal: Let $d(X_k,j)$ be the minimum distance between any one of the points $x_1 \ldots x_k$ and the node j on G. The goal is to minimize

$$J(X_k) = \sum_{i=1}^{n} h_j d(X_k, i)$$

k-median Problem - Hakimi's theorem [1964].

Theorem: At least one optimal set of k-medians exist solely on the nodes of G.

Practical significance: the search for the set of the k optimal locations for the k facilities can be limited to the node set of G instead of the infinite number of points that lie on the links of G.

Proof: In class

1-Median Problem – *optimal algorithm*

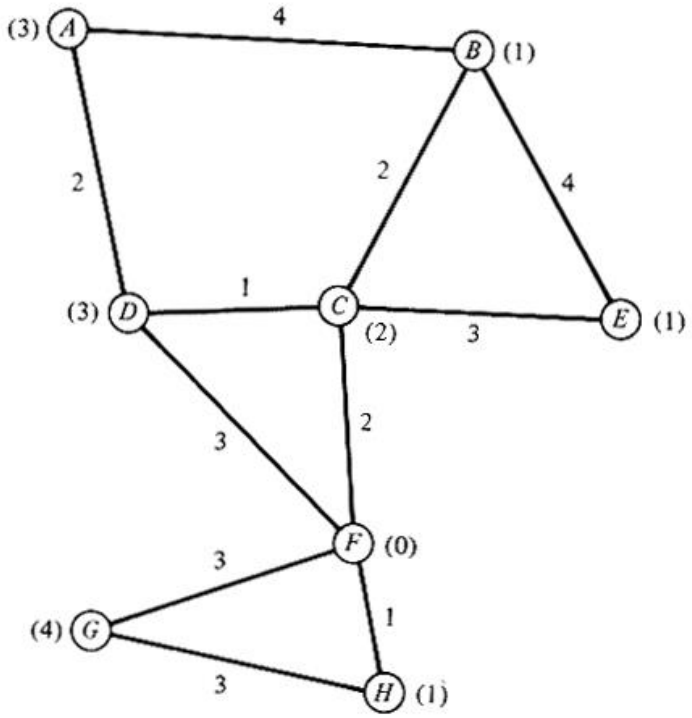
STEP 1: Obtain the minimum distance matrix for the nodes of *G*.

STEP 2: Multiply the j^{th} column of the minimum distance matrix by the demand weight h_j (j = 1, . . ., n) to obtain the matrix $h_j \cdot d(i,j)]$.

STEP 3: For each row i of the $[h_j \cdot d(i\ j)]$ matrix, compute the sum of all the terms in the row. The node that corresponds to the row with the minimum sum of terms is the location for the 1-median.

1-Median Problem - *Example*

Input: The weighted Network



Step 1: calculate shortest path matrix

From \ To	A	B	C	D	E	F	G	H
A	0	4	3	2	6	5	8	6
B	4	0	2	3	4	4	7	5
C	3	2	0	1	3	2	5	3
D	2	3	1	0	4	3	6	4
E	6	4	3	4	0	5	8	6
F	5	4	2	3	5	0	3	1
G	8	7	5	6	8	3	0	3
H	6	5	3	4	6	1	3	0

$= [d(i,j)]$

1-Median Problem - *Example*

Step 2: compute $h_j \cdot d(i, j)$ by multiplying each column of the distance matrix by the weight of node j.

$[h_j d(i,j)] =$

	A	B	C	D	E	F	G	H
A	0	4	6	6	6	0	32	6
B	12	0	4	9	4	0	28	5
C	9	2	0	3	3	0	20	3
D	6	3	2	0	4	0	24	4
E	18	4	6	12	0	0	32	6
F	15	4	4	9	5	0	12	1
G	24	7	10	18	8	0	0	3
H	18	5	6	12	6	0	12	0

Step 3: Pick the optimal node.

Facility located at	A	B	C	D	E	F	G	H
Total travel distance	60	62	40	43	78	50	70	59
Average travel distance $\left(=\frac{\text{total travel distance}}{15}\right)$	4.0	4.13	2.67	2.87	5.2	3.33	4.67	3.93

k-Median Problem - *Example*

Assume that two facilities were desired (2-median).

We can still take advantage of Hakimi's theorem and consider only sets of points composed of two nodes. There are $\binom{n}{2} = \binom{8}{2} = 28$ possibilities.

Total distances for each combination of locations can still be calculated from the $[h_j d(i,j)]$ matrix

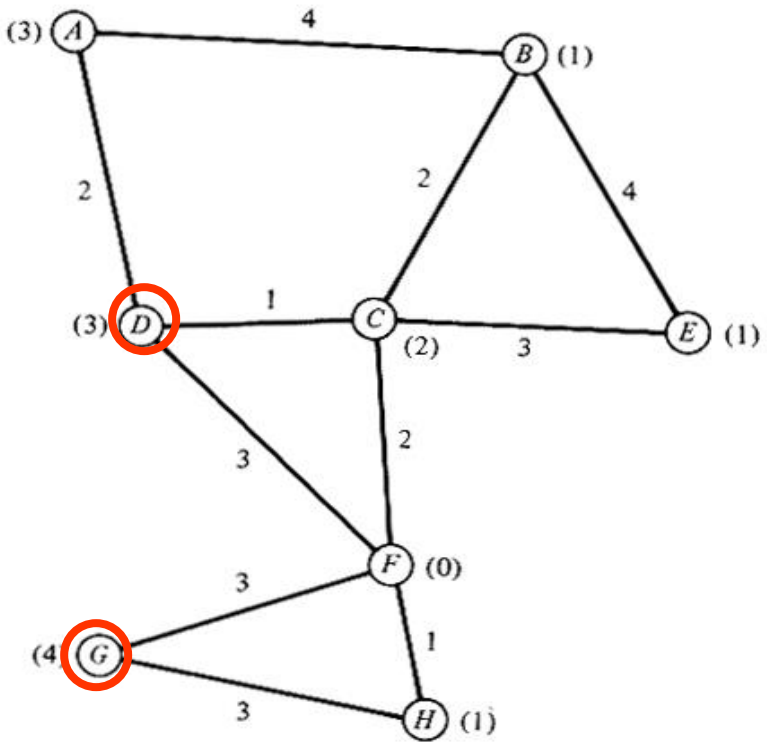
For instance, if the two facilities are A,G, the amount of travel contributed by users from D is given by Min $[h_D d(A,D), h_D\ d(G,D)]$ = Min [6, 18] = 6

k-Median Problem - *Example*

By exhaustive consideration of all possibilities, it turns out that {D,G} is an optimal solution.

Note: the total distance is 18 units.

- A, B. C, D, and E are assigned to D; G and H are assigned to G → unbalanced.
- Clients from B. C, and E now have to travel farther than they had to with a single facility.



k-Median Problem

For larger k, this approach works but is non-efficient.

There are $\binom{n}{k}$ possible choices of medians.

Polynomial for fixed k, but not so practical.

Non-poly for $k=\Omega(n)$

There are several good known practical heuristics for this problem.

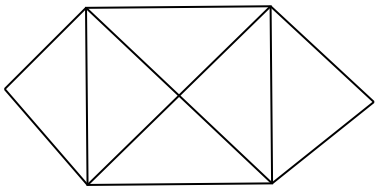
Routing Problems

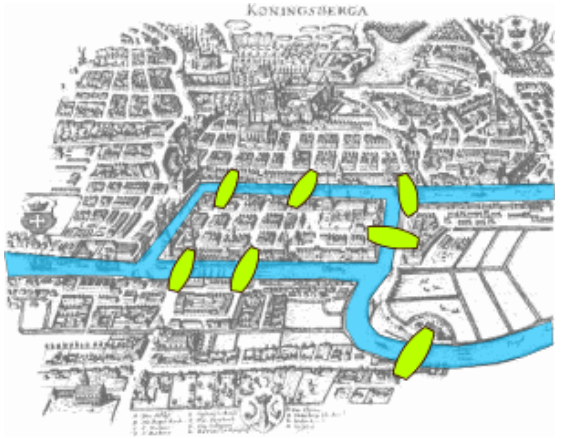
- A single (or a set of) vehicle(s) must travel from a single (or a set of) depot(s) to a set of customers to make a delivery (and back to depot). Route the vehicles so that all demand is met and total distance (other objectives) is minimized.
- There are many variants, with many applications. We are going to see two problems:

1. The Chinese Postman Problem
2. Traveling Salesman Problem

Review: Euler path

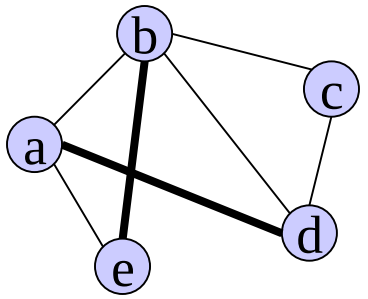
- "draw without lifting your pen from the paper".
- An Euler path in a graph is a path in the graph that visits each edge exactly once. An Euler cycle begins and ends at the same vertex.
- Well known that an undirected graph has an Euler cycle iff (1) it is connected and (2) each vertex has an even degree.
- There exists a simple algorithm for constructing an Euler cycle.





Review: Matching

- a matching in a graph G=(V,E): a set of edges S from E such that each vertex in V is incident to at most one edge of S.
- Example: M={(a,d),(b,e)} is a matching.
  S={(a,d), (c,d)} is not a matching.



- A maximum matching in G: a matching of maximum cardinality
- A perfect matching: a matching of size n/2
- A minimum weight max-matching in a weighted graph: a maximum matching of minimum total weight. Can be found in poly-time.

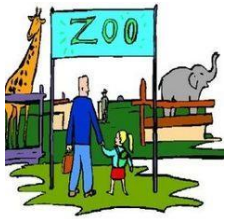
The Chinese Postman Problem

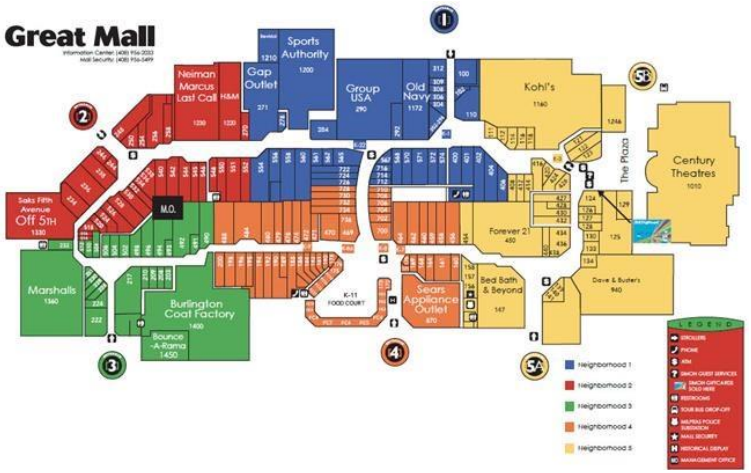
Input: a connected weighted undirected graph G.

Output: a shortest path (or cycle) that includes each edge at least once.

Many Real-life Applications:

- Delivery
- Site seeing
- Maintenance :Trash collection, road sweeping, snow-plows.
- Patrolling





The Chinese Postman Problem

For a graph $G=(V,E)$, let $w(E)=\Sigma_{e\in E} w(e)$.

We are going to consider the shortest-cycle problem (it is easy to move to the shortest-path variant).

Simple observations:

1. $OPT \geq w(E)$.
2. $OPT = w(E)$ iff G has an Euler cycle.
3. $OPT > w(E)$ if some edges have odd degrees.

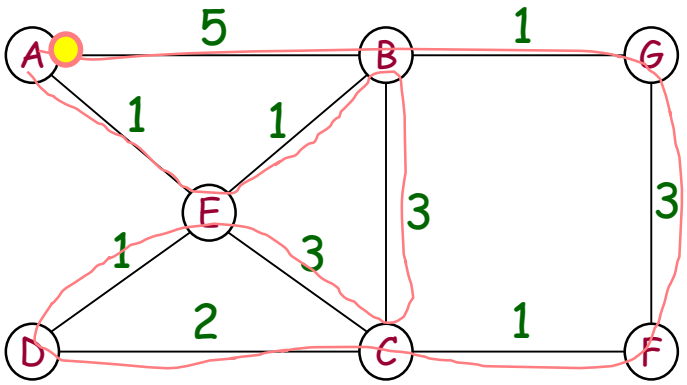
Claim: In every graph, there exists an even number of vertices having an odd degree.

Proof: In class

The Chinese Postman Problem

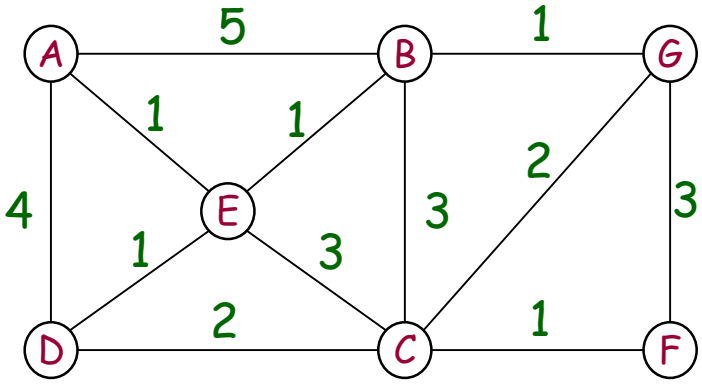
Example 1: This graph has an Euler cycle.

OPT = w(E)= 21,
p=ABGFCDECBEA.



Example 2: This graph has no Euler cycle.

An optimal solution will have to pass several edges more than once.



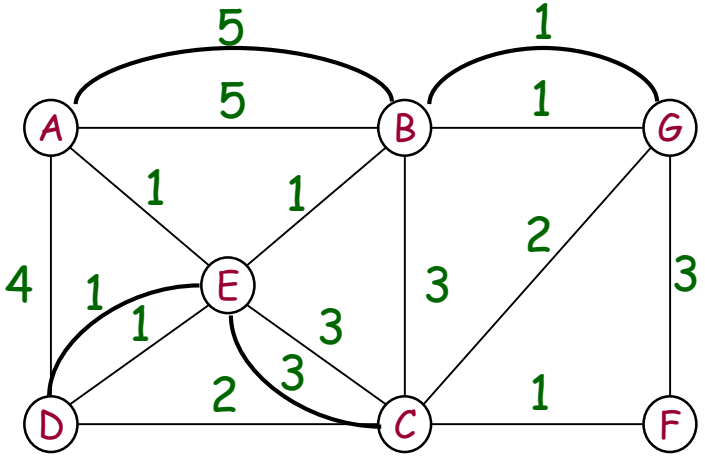
The Chinese Postman Problem

Definitions:

- An extension of G: a graph G' obtained from G by replacing each edge $e \in E$ by one or more parallel edges (having the same weight).
- An Eulerian extension of G: an extension of G, in which each vertex has an even degree.

Lemma 1: G has a postman cycle of length k if and only if G has an Eulerian extension of total edge weight k.

Proof: In class



Example: postman cycle of length 28+10.

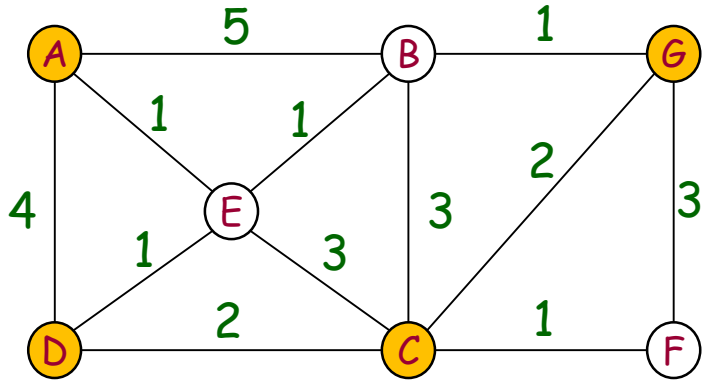
The Chinese Postman Problem – An optimal algorithm by Edmonds and Johnson (1973)

1. Let $S = \{v_1, v_2, \ldots, v_{2m}\}$ be the set of vertices of odd degree in G. For each $v_i, v_j \in S$ let $p_{i,j}$ be a shortest path connecting v_i and v_j in G.
2. Construct a weighted complete graph K_{2m} with $V(K_{2m}) = S$ and $w(v_i,v_j) = \text{length}(p_{i,j})$ for all $v_i,v_j \in S$.
3. Find a minimum weight perfect matching M in K_{2m}.
4. Construct G^* - an Eulerian extension of G, by replacing, for all $(v_i,v_j) \in M$, each edge $e \in p_{i,j}$ by two parallel edges.
5. An optimal postman cycle is an Euler cycle in G^*

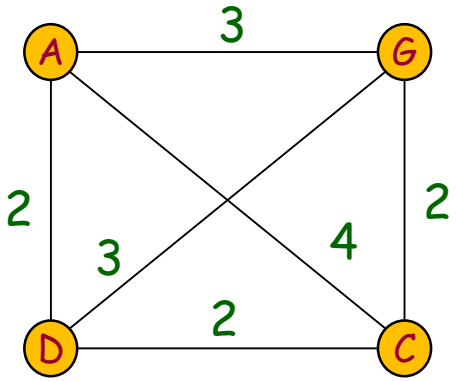
The Chinese Postman Problem

Example:

S={A,C,D,G}

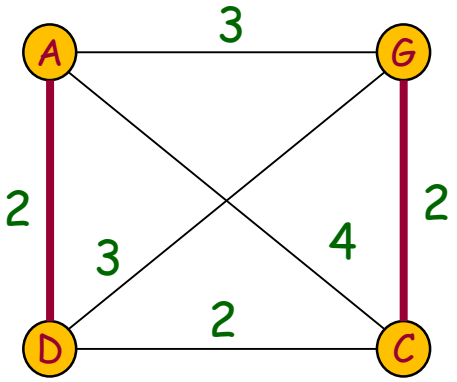


K_4 with V=S and $w(v_i,v_j)$= shortest path in G.



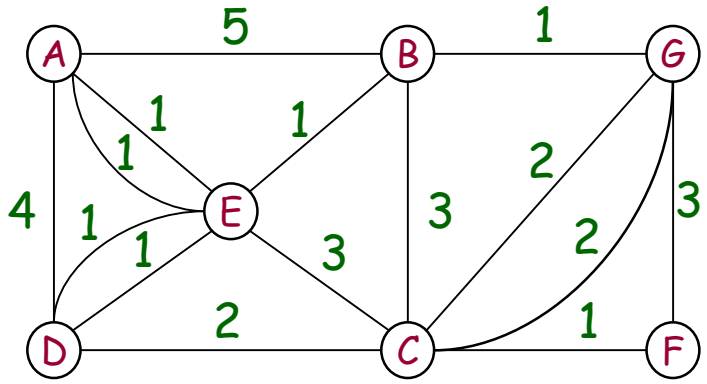
The Chinese Postman Problem

A Minimum weight perfect matching is M = {AD, CG}.
w(M) = 2 + 2 = 4.



Back to G:
p_{AD} = A-E-D
p_{CG} = C-G

Extend G to an Eulerian graph accordingly.



OPT: ABCDAEBGCGFCEDEA
w(OPT) = w(E) +w(M)= 28+4=32.

The Chinese Postman Problem

Theorem: Edmonds-Johnson Algorithm outputs a shortest postman cycle in G.
Proof: By Lemma 1, it suffices to show that G^* is a minimum weight Eulerian extension of G.

Lemma: Assume that G has $2m$ vertices of odd degree. Then G has a set of m pairwise edge-disjoint paths P such that each odd-degree vertex in G is an end vertex of exactly one path in P.

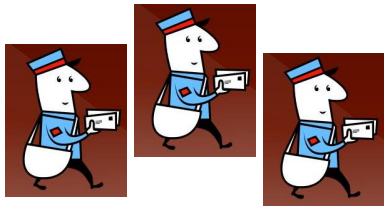
Proof and wrap-up: in class.

The Chinese Postman Problem

Variants:

k-CPP: there are k postmen that need to patrol the whole network.

Possible objective:



-Minimal total length of tours.

-Minimal maximal length of tour.

-Minimal (total or max) deviation from a deadline.

Rural Postman problem: Only a subset of the edges must be traversed (other edges may be included in the path)

Windy postman problem: The graph is undirected, but the traversal time depends on the direction.

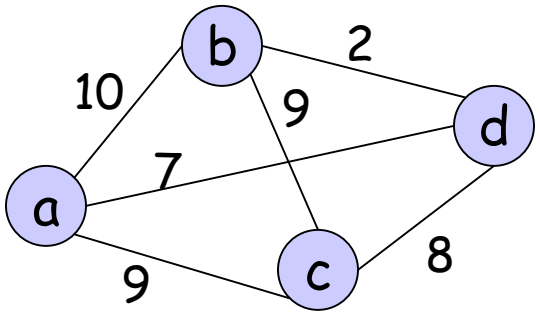
All the above variants are NP-hard (k-CPP is NPH even for constant k).

The Traveling Salesman Problem

The Problem: Given n points in the plane (corresponding to locations of n cities) find a shortest traveling salesman tour – that passes exactly once in each of the points.

For each pair of cities a,b, we are given the distance dist(a,b) from a to b.

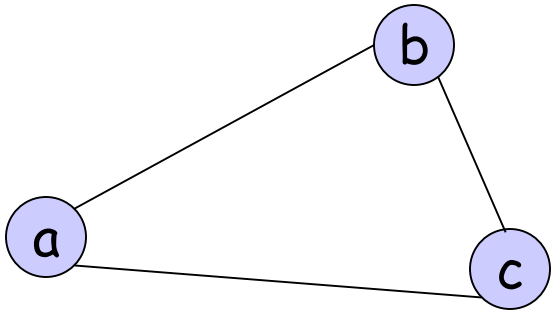
In other words, the input is given as a weighted complete graph.



Euclidean Traveling Salesman Problem

Distances in the plane satisfy the triangle inequality:
dist(a,b) ≤ dist(a,c) + dist(c,b)

It means that direct routes are always shorter than indirect routes.



For this version, we will see a simple 2-approximation algorithm: we will find in poly-time a tour whose length is at most twice the optimal.

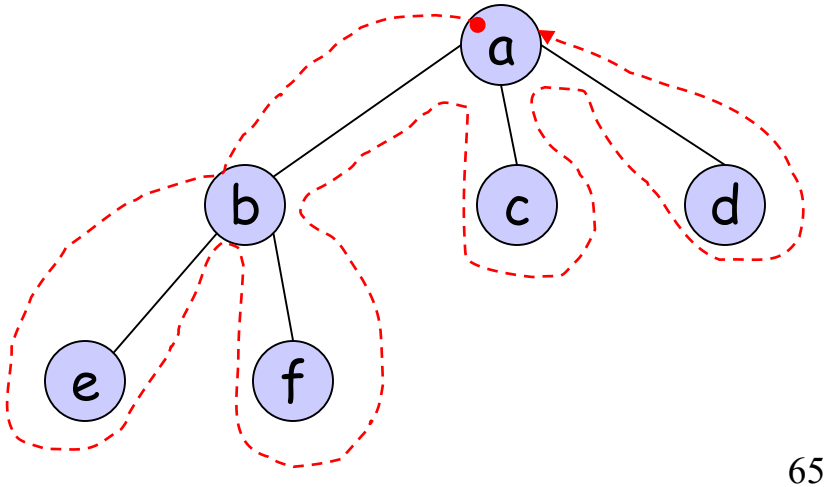
Approximating Euclidean TSP

Note: The weight of a minimum spanning tree is always less than the weight of the optimal tour.

Why? because by removing any edge from the optimal tour we get a spanning tree.

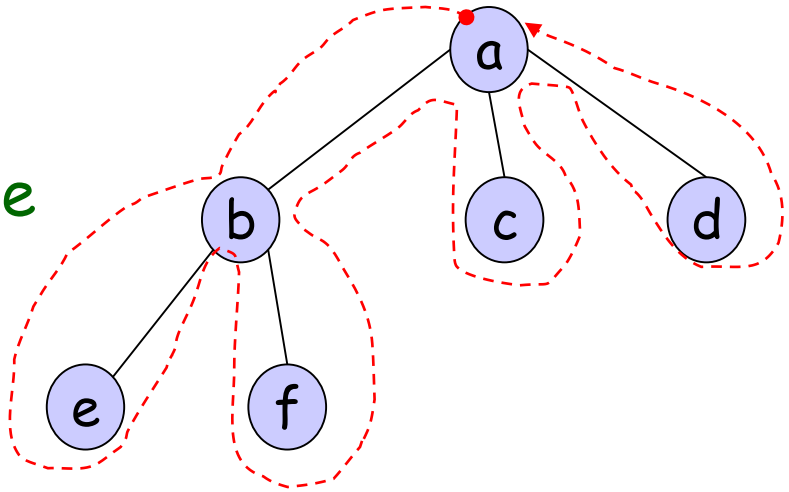
We will use this property to obtain an approximate solution.

Assume that this is our MST. Consider a DFS tour on this tree, let it be (w.l.o.g) a-b-e-b-f-b-a-c-a-d-a

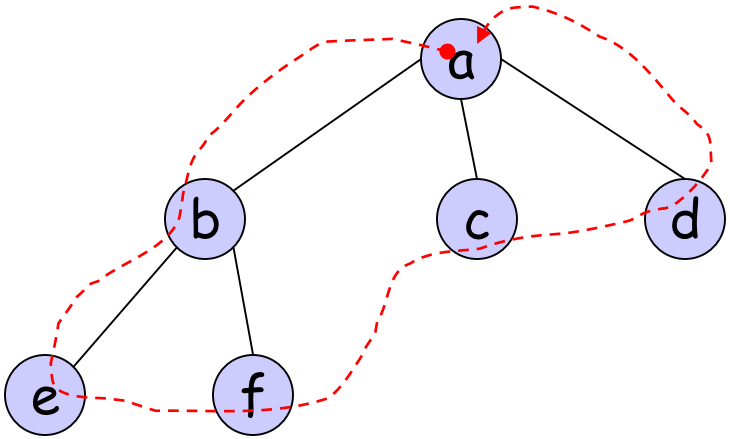


Approximating Euclidean TSP

- The DFS tour defines a TSP path that visits all the cities. The length of this tour is twice the total weight of the MST. However, we might visit some cities more than once.



- To get a legal solution we make shortcuts (move in the next step to the next unvisited vertex). This can only reduce the total length of the path.



Approximating Euclidean TSP

The resulting algorithm:

1. Find a minimum spanning tree of points
2. Convert to tour by following DFS and including edge in opposite direction when DFS backtracks.
3. Construct shortcuts by taking direct routes instead of backtracking.

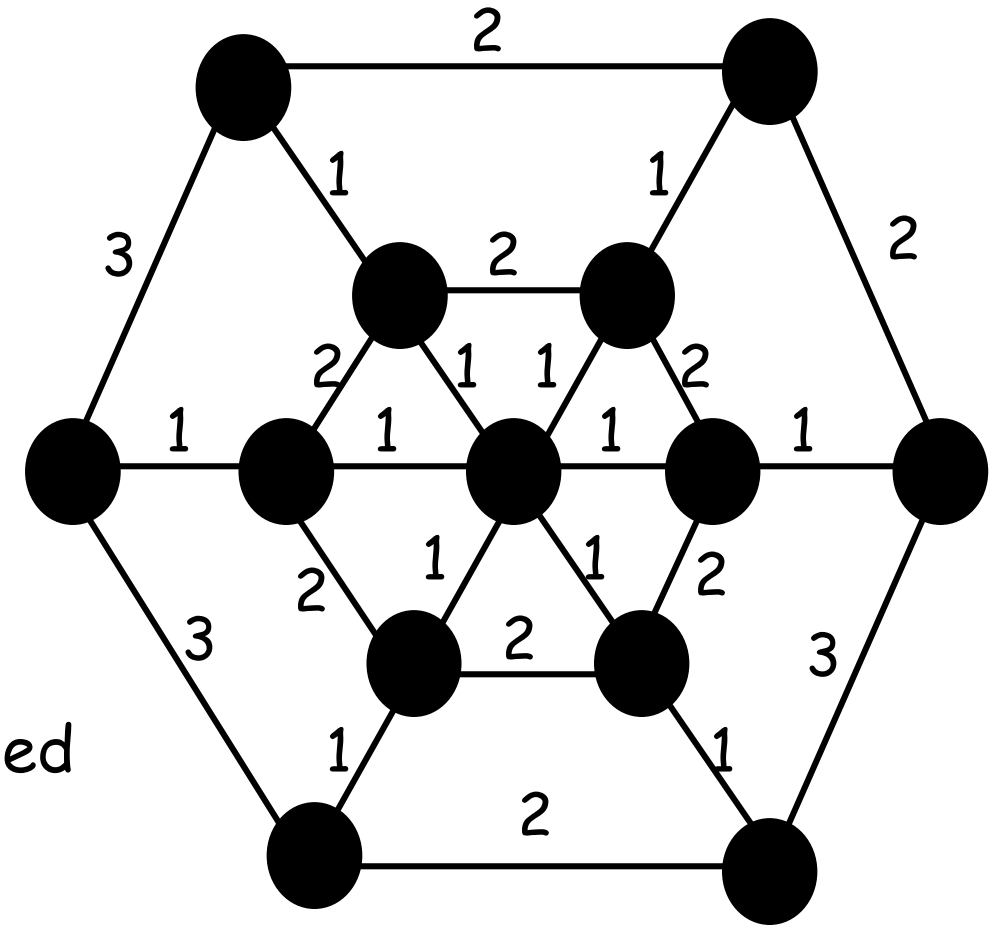
The length of the resulting tour is at most 2 times the optimal → this is a 2-approximation algorithm.

Christofides' Algorithm

An improvement of the 2-approximation algorithm. Based on combining additional tools (Euler cycle, matching).

1. Find an MST.
2. Find minimum weight matching of odd-degree vertices in the tree (there's an even number of them).
3. Find an Eulerian tour of MST plus edges in matching.
4. Make shortcuts.

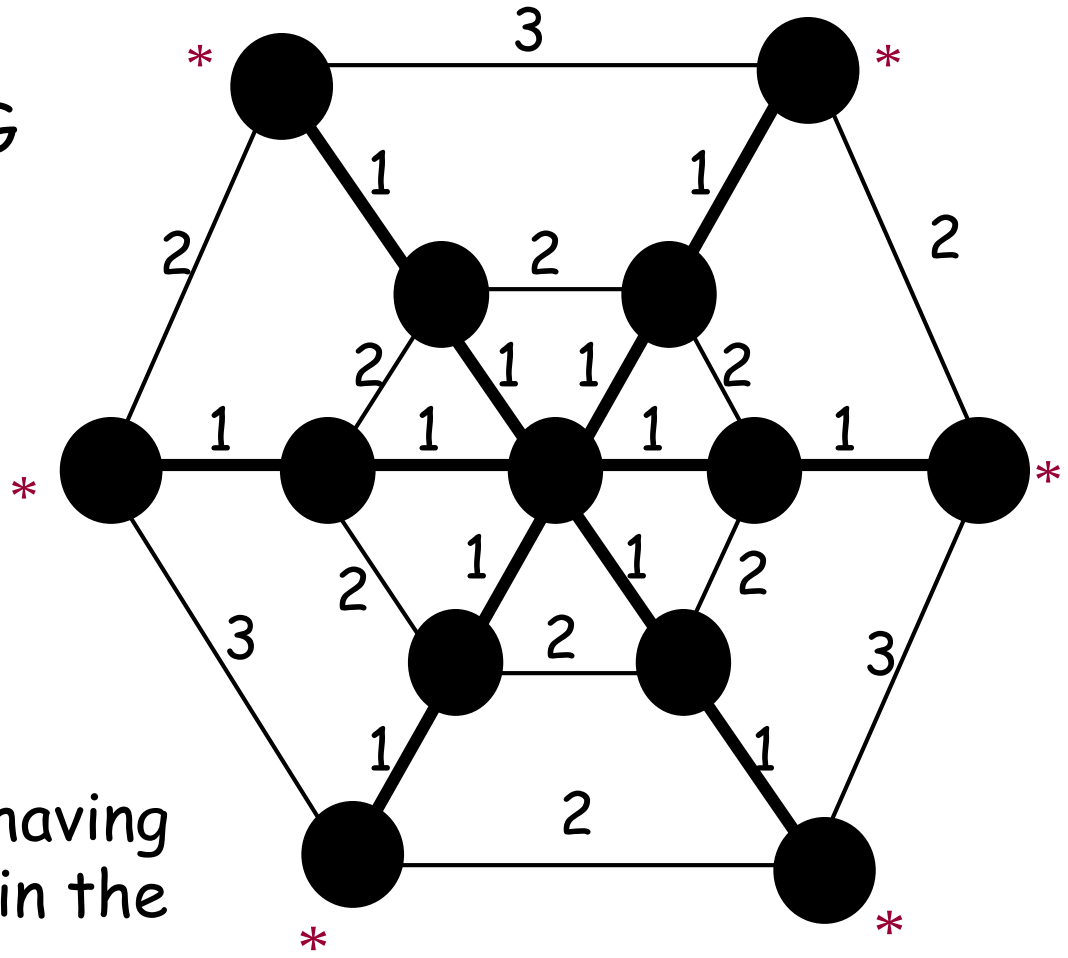
Approximating TSP, Example (1)



The weighted graph G.

Approximating TSP, Example (2)

An MST in G

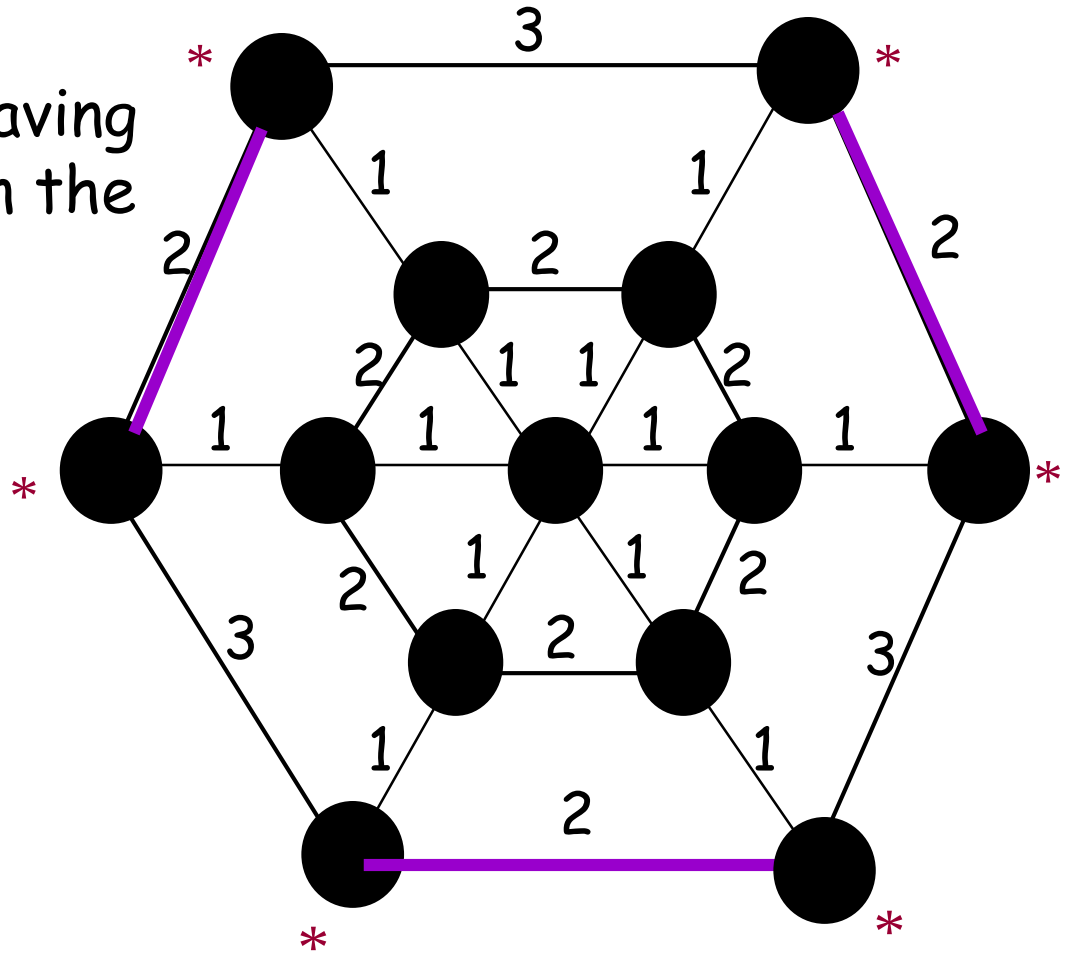


*- vertices having odd degree in the MST

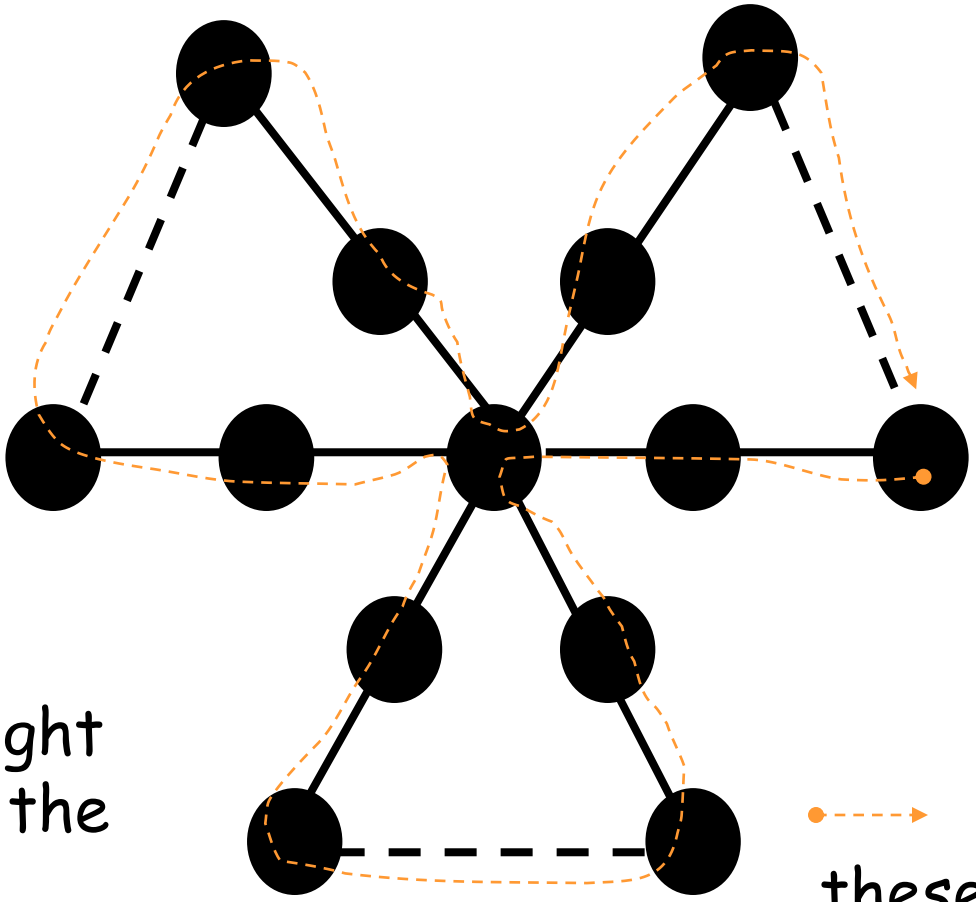
Approximating TSP, Example (3)

*- vertices having odd degree in the MST

A minimum weight matching of the * vertices.



Approximating TSP, Example (4)



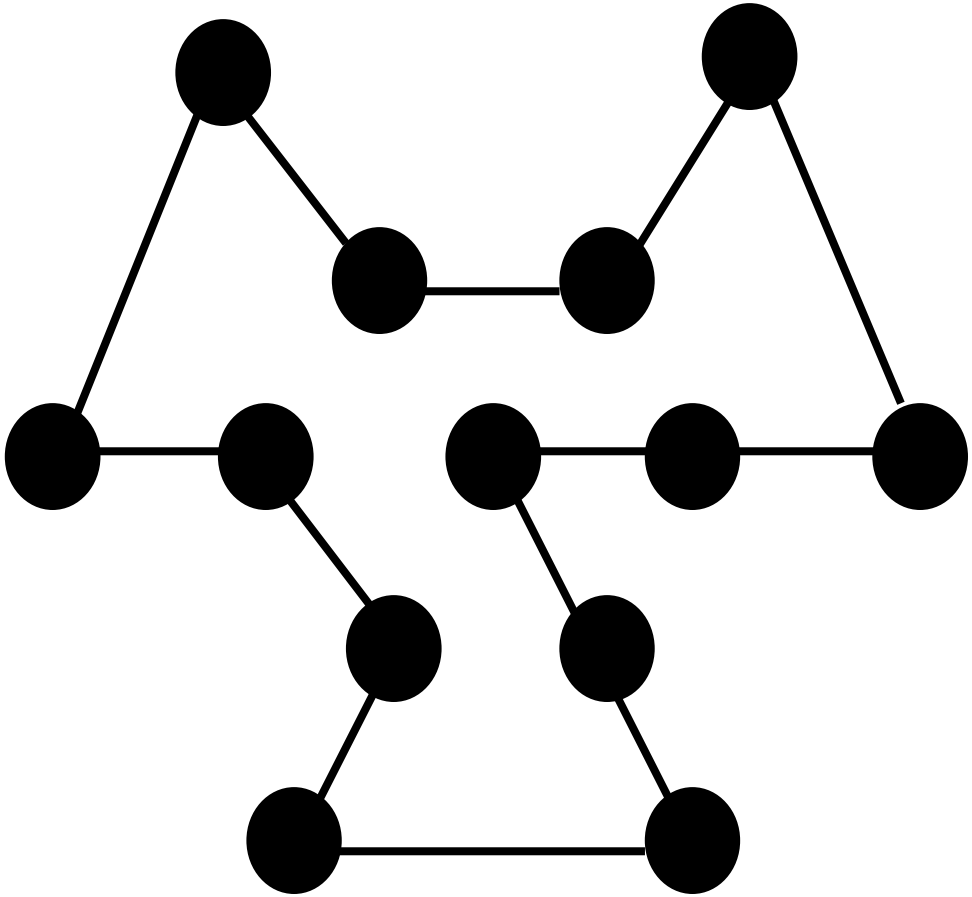
The MST + minimum weight matching of the odd-degree vertices.

Euler tour on these edges

Approximating TSP, Example (5)

Apply shortcuts.

The final tour:



Analysis of Christofides' Algorithm

Theorem: The approximation algorithm for Euclidean TSP finds a tour of length at most 3/2 optimal.

Proof:

- weight of MST $\leq$ weight of optimal tour
- weight of matching $\leq$ (weight of optimal tour)/2 (next slide)
- shortcuts don't cost

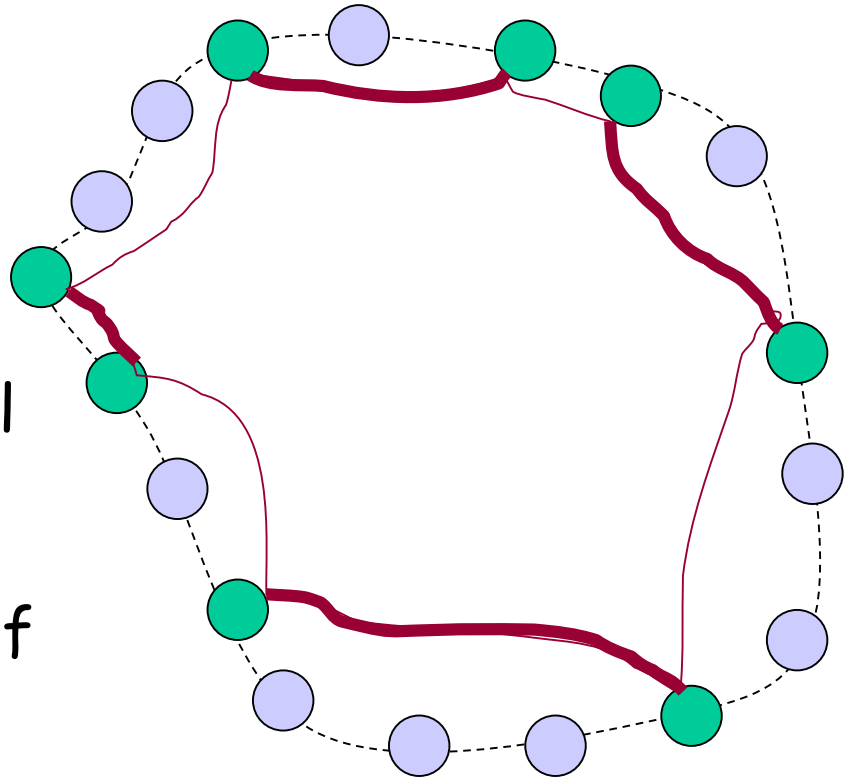
Analysis of Christofides' Algorithm

green vertices: odd-degree vertices in MST. (their # is even)

Any Optimal TSP tour (dotted line) visits these vertices in some order.

In red: Shortcuts of optimal TSP through these vertices.

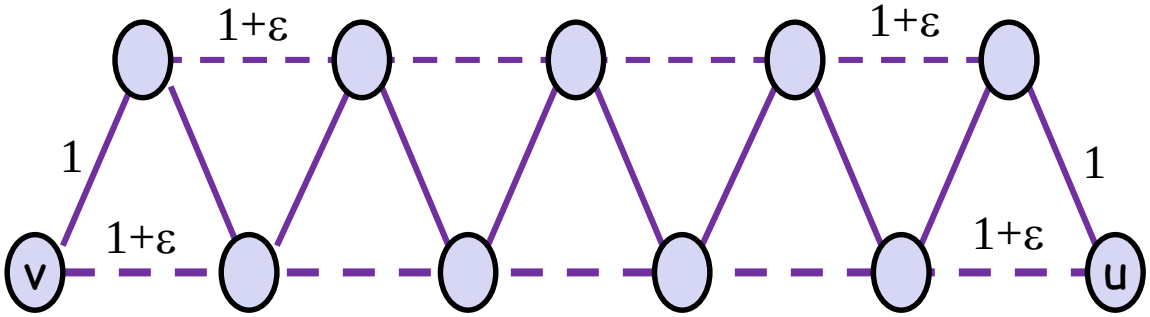
In **bold** red: The light half of the shortcuts.



A minimum weight matching is lighter than the **bold** shortcuts $\leq \frac{1}{2}$ optimal TSP.

Analysis of Christofides' Algorithm

The 3/2-analysis is tight:



There are 2n+1 nodes, all solid edges have weight 1, all dashed edges have weight 1+ε.
Other edges – max weight satisfying the triangle inequality (for example, (u,v) has cost (n+1)(1+ε)).

Alg's tour: ~3n. OPT = ~2n (in class).

Non-Euclidean TSP

When did we use the assumption that for all a,b,c $dist(a,b) \leq dist(a,c) + dist(c,b)$?

Is it really needed?

Yes – get ready for our first non-approximability result!

Theorem: For any constant c, if there is a polynomial time c-approximation algorithm for TSP then P=NP.

Non-Euclidean TSP

Proof: Reduction from the Hamiltonian cycle problem. Assume by contradiction that there exists an algorithm A that gets a TSP instance G' and returns a TS tour whose cost is at most c times the optimal.

In particular, if G' has a TS tour of cost n, then A finds a tour of cost at most cn.

We will use algorithm A to solve the Hamiltonian-cycle decision problem.

Non-Euclidean TSP

Given G=(V,E) and the question "Is there a HC in G"
we construct a TSP instance G', such that there is a HC in G if there is a TS tour of cost at most n in G' and there is no HC in G if the minimal TS tour has cost > cn.

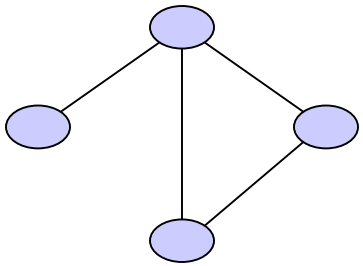
G'=(V,E') is a clique.

The weight of edges in E is 1

The weight of any additional edge is c·n.

- A HC in G corresponds to a TSP tour of weight n.
- Any tour that uses one or more additional edges has cost > cn.

G



G'