# Concepts of approximation algorithms

**Optimization Problem:**

The solution of the problem is associated with a cost (value).

We want to ***maximize*** the cost or ***minimize*** the cost.

The Traveling Salesman Problem and shortest path are optimization problems.

Hamiltonian Path problem is NOT an optimization problem (it is a *decision* problem).

# Approximation algorithms

An algorithm *A* is an approximation algorithm , if given any instance *I,* it finds a candidate solution *s(I).*

How good an approximation algorithm is?

We use *performance ratio* to measure the quality of an approximation algorithm.

# Approximation algorithms

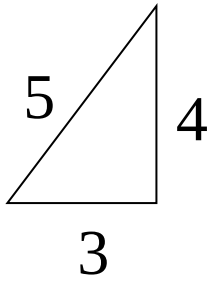
For a minimization problem, the performance ratio of algorithm A is defined as a number *r* such that for any instance I of the problem, A(I) / OPT(I) is at most r (r >= 1), where OPT(I) is the value of the optimal solution for instance I and A(I) is the value of the solution returned by algorithm A on instance I.

For maximization problem, the performance ratio of algorithm A is defined as a number r such that for any instance I of the problem, OPT(I) / A(I) is at most r (r>=1), where OPT(I) is the value of the optimal solution for instance I and A(I) is the value of the solution returned by algorithm A on instance I.

# Approximation problem example: TSP

A very natural restriction of the TSP is the triangle inequality:

For any 3 cities A, B and C, the distance between A and C must be at most the distance from A to B plus the distance from B to C.




Most natural instances of TSP satisfy this constraint.

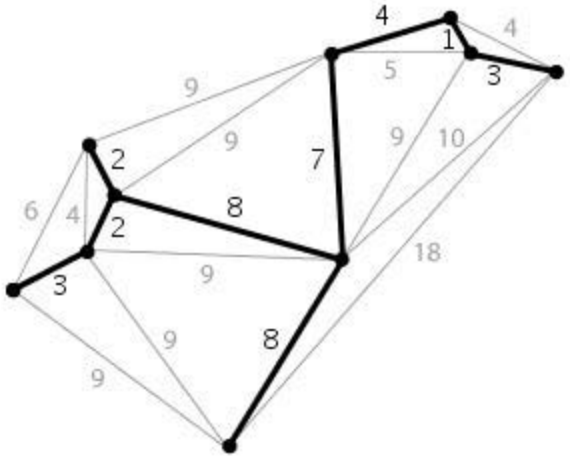
In this case, there is a constant-factor approximation algorithm (due to Christofides, 1975) which always finds a tour of length at most 1.5 times the shortest tour.

We explain a weaker (but simpler) algorithm which finds a tour of length at most twice the shortest tour.

# Approximation problem example: TSP

For this approximation algorithm, we will use the notion of a Minimum Spanning Tree.

Given a connected, undirected graph, a spanning tree of that graph is a subgraph which is a tree and connects all the vertices together. A single graph can have many different spanning trees. We can also assign a *weight* to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A **minimum spanning tree** or **minimum weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree.



# Minimum Spanning Tree history

The Minimum Spanning Tree problem is interesting in its own merit:

The first algorithm for finding a minimum spanning tree was developed by Czech scientist Otakar Borůvka in 1926 (see Boruvka's algorithm). Its purpose was an efficient electrical coverage of Bohemia.

There are now two algorithms commonly used, Prim's algorithm and Kruskal's algorithm. All three are greedy algorithms that run in polynomial time, so the problem of finding such trees is in **P**.

The fastest minimum spanning tree algorithm to date was developed by Bernard Chazelle, and based on Borůvka's. Its running time is $O(e\ \alpha(e,v))$, where $e$ is the number of edges, $v$ refers to the number of vertices and $\alpha$ is the classical functional inverse of the Ackermann function. The function $\alpha$ grows extremely slowly, so that for all practical purposes it may be considered a constant no greater than 4; thus Chazelle's algorithm takes very close to $O(e)$ time.

# Minimum Spanning Tree history

What is the fastest possible algorithm for this problem? That is one of the oldest open questions in computer science. There is clearly a linear lower bound, since we must at least examine all the weights. If the edge weights are integers with a bounded bit length, then deterministic algorithms are known with linear running time, $O(e)$.

For general weights, David Karger exhibited a randomized algorithm whose *expected* runtime is linear.

Whether there exists a deterministic algorithm with linear running time for general weights is still an open question.

Just for reference, the algorithms of Kruskal and Prim run in time O(e log(v)) (Prim's algorithm complexity depends on data structures used), which for practical purposes is adequate.

Question: Why is O(e log(v)) equal to O(e log(e))?

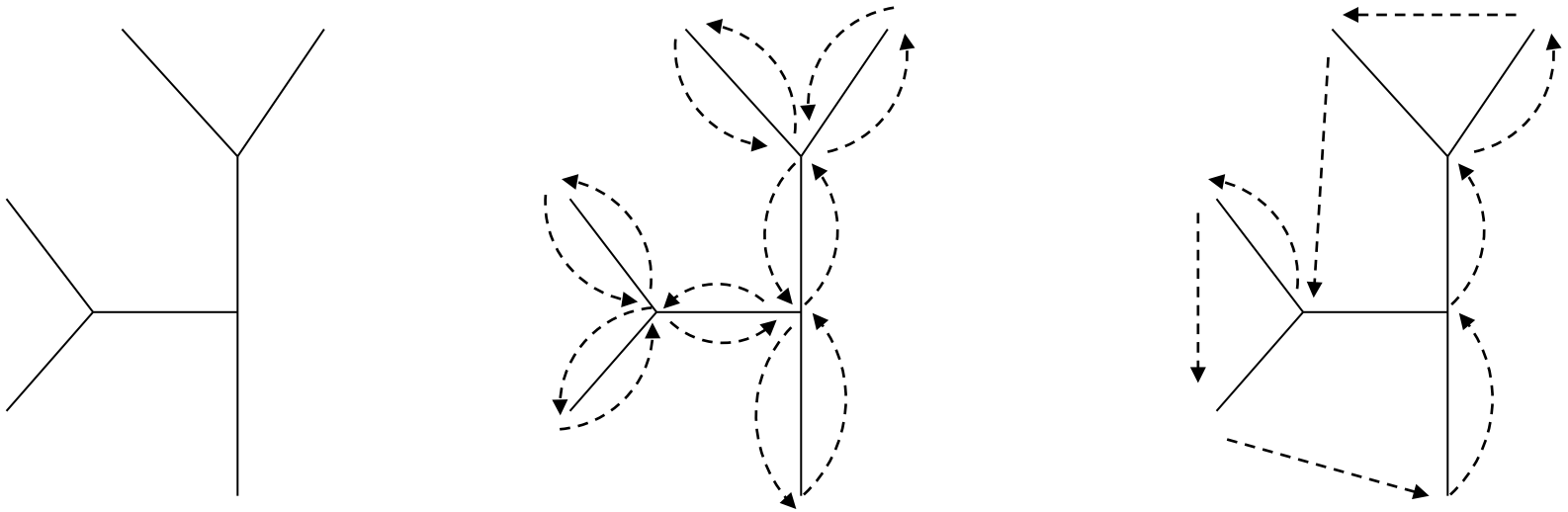
# Approximation problem example: TSP

The length of the minimum spanning tree of the network is a natural lower bound for the length of the optimal route. Why?

In the TSP with triangle inequality case it is possible to prove upper bounds in terms of the minimum spanning tree and design an algorithm that has a provable upper bound on the length of the route. The algorithm is the following:

1. Construct the minimum spanning tree.
2. Duplicate all its edges. That is, wherever there is an edge from *u* to *v*, add a second edge from *u* to *v*. This gives us an Eulerian graph.
3. Find a Eulerian cycle in it. Clearly, its length is twice the length of the tree.
4. Convert the Eulerian cycle into the Hamiltonian one in the following way: walk along the Eulerian cycle, and each time you are about to come into an already visited vertex, skip it and try to go to the next one (along the Eulerian cycle).

# Approximation problem example: TSP



(a)
A spanning tree

(b)
Twice around the tree

(c)
A tour with shortcut