République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

Faculty of exact sciences and computer science
Jijel University

Practical Assignment

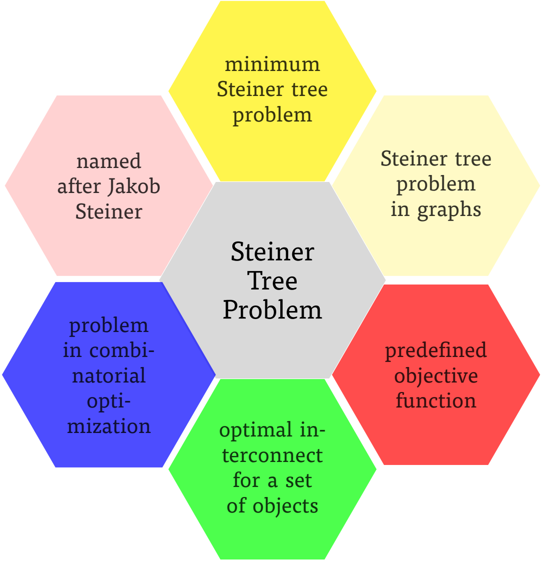
Resolution of Steiner Tree Problem

For:

- IA class → TAIA1 module
- RS class → APG module

Presented by:
Dr. Hamida BOUAZIZ

Steiner tree problem

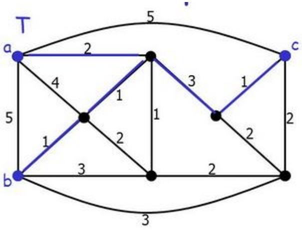


Definition of Steiner tree problem in graphs

Given an undirected graph with:

- non-negative edge weights, and
- a subset of vertices, usually referred to as **terminals**

The Steiner tree problem in graphs requires a tree of **minimum weight** that contains **all terminals** (but may include additional vertices)



Applications: The Steiner tree problem in graphs has applications in:

- circuit layout,
- network design.

Steiner tree problem execution time

The Steiner tree problem in graphs can be seen as a generalization of two other famous combinatorial optimization problems:

- the **(non-negative) shortest path problem**. If a Steiner tree problem in graphs contains exactly two terminals, it reduces to finding the shortest path.

- the **minimum spanning tree problem**. If, on the other hand, all vertices are terminals, the Steiner tree problem in graphs is equivalent to the minimum spanning tree.

Spanning Tree?

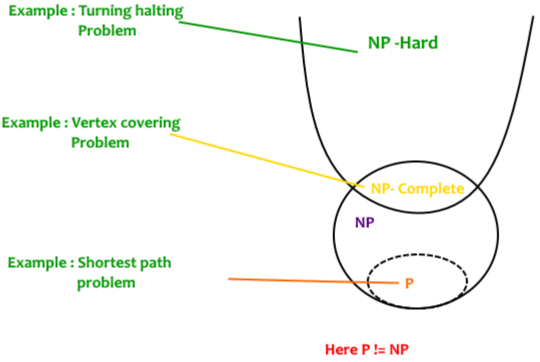
Given an undirected and connected graph G= $\langle$ V,E $\rangle$, a spanning tree of the graph G is a tree that spans G (that is, it includes **every vertex of G**) and is a sub-graph of G(every edge in the tree belongs to G).

While both the non-negative shortest path and the minimum spanning tree problem are solvable in **polynomial time**, the decision variant of the Steiner tree problem in graphs is **NP-complete**.

Why is it NP-Complete?

NP-complete problems are the hardest problems in the NP (Non-deterministic Polynomial time) set. A decision problem L is NP-complete if:

1. L is in NP (Any given solution for NP-complete problems can be verified quickly, but there is no efficient known solution).
2. Every problem in NP is reducible to L in polynomial time.



Decision vs.optimization problems

- NP-completeness applies to the realm of **decision problems**. It was set up this way because it is **easier to compare the difficulty** of decision problems than that of optimization problems.

- By being able to solve a decision problem in polynomial time will often permit us to solve the corresponding optimization problem in polynomial time.

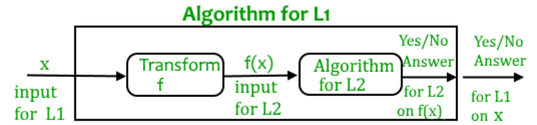
Example

- Consider **the vertex cover problem** (Given a graph, find out the minimum sized vertex set that covers all edges). It is an **optimization problem**.

- Corresponding decision problem is, given undirected graph G and k, is there a vertex cover of size k?

What is Reduction?

Let L1 and L2 be two decision problems. Suppose algorithm A2 solves L2. The idea is to find a transformation from L1 to L2 so that algorithm A2 can be part of an algorithm A1 to solve L1.

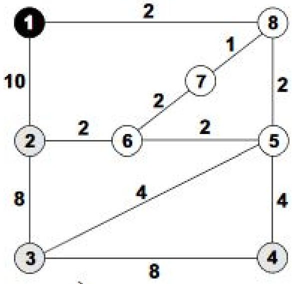


KMB Heuristic to solve Steiner Tree Problem

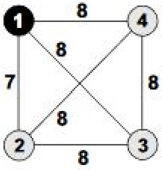
- **K**ou, **M**arkovsky and **B**ermann proposed such KMB minimum Steiner tree heuristic.

- For an undirected graph N= ⟨ V, E ⟩ (see Fig. a) and a set of **Terminal nodes** G (in the example below, Terminals are {1,2,3,4}):

  - Construct **complete undirected graph** N1= (V1, E1), constructed from Terminal nodes G (paths in N1 are **shortest paths in N**) (see Fig. b).



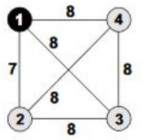
a)



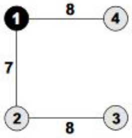
b)

KMB Heuristic

- Find the **minimum spanning tree T1** for graph **G1** (if there are several minimum spanning trees, pick an arbitrary one) (see Fig. c)

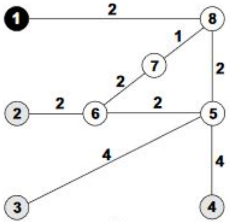


b)



c)

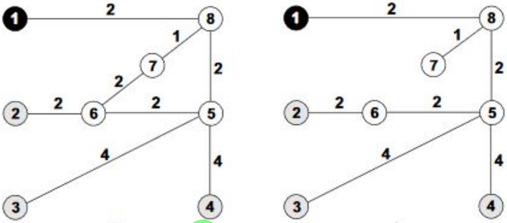
- Construct the sub-graph $\mathbf{G}_S$ of G by replacing each edge in T1 by corresponding shortest path in G (see Fig. d). If there are several shortest path, pick an arbitrary one.



d)

KMB Heuristic

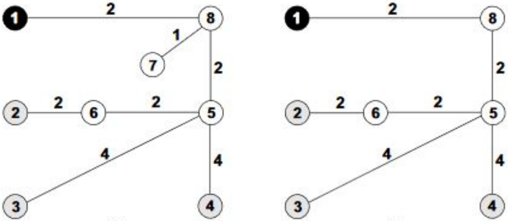
- **Find the minimal spanning tree** T_S of G_S (see Fig. e). If there are several minimum spanning trees,pick an arbitrary one.



d) e)

cost 16

- **Construct a Steiner tree** T_{KMB} from T_S by deleting edges in T_S, if necessary, so that all the leaves in T_{KMB} are Steiner points (see Fig. f).



e) f)

References for KMB heuristic

[1] Piechowiak, M., Zwierzykowski, P., & Hanczewski, S. (2005, July). Performance analysis of multicast heuristic algorithms. In Third International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks. Networks UK Publishers.

Which algorithm may we use to build the spanning tree ?

There are a lot of these algorithms, such as: reverse-delete algorithm, Kruskal's algorithm, Prim's algorithm, Boruvka's algorithm, etc.

In this class, we are interested in

Kruskal's algorithm

Kruskal's algorithm

- Kruskal's algorithm is a **minimum-spanning-tree** algorithm.
- It is a **greedy algorithm** in graph theory as it finds a minimum spanning tree for a connected weighted graph by adding increasing cost arcs at each step.
- This means it finds a subset of the edges that forms a tree that includes **every vertex**, where the total weight of all the edges in the tree is minimized.
- If the graph is not connected, then it finds a **minimum spanning forest** (a minimum spanning tree for each connected component).

Kruskal's Algorithm Steps

- Create a forest F (a set of trees), where each vertex in the graph is a separate tree.
- Create a set S containing all the edges in the graph
- **while S is nonempty and F is not yet spanning**
  - Remove an edge with minimum weight from S
  - If the removed edge connects two different trees then add it to the forest F, combining two trees into a single tree
- At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree

Pseudocode of Kruskal's Algorithm

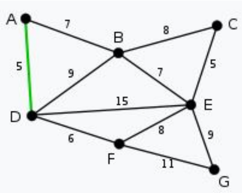
- A: represents the set of the minimum spanning tree edges.
- V: are the vertices of G.
- E: are the edges of G, they are ordered form the least to the most costly.

KRUSKAL(G):

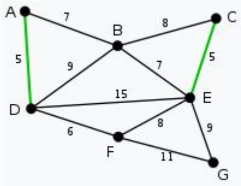
```
A ← ∅
for each v ∈ V do
    MAKE-SET(v)
for each (u, v) in E do
    if FIND-SET(u) ≠ FIND-SET(v) then
        A ← A ∪ {(u, v)}
        UNION(FIND-SET(u), FIND-SET(v))
return A
```

Example of Applying Kruskal's Algorithm

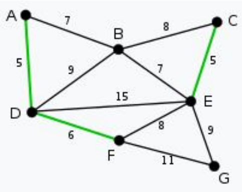
step 1



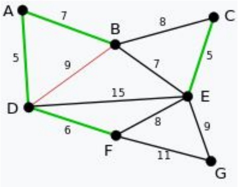
step 2



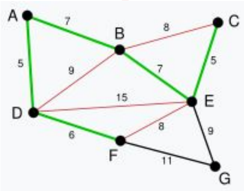
step 3



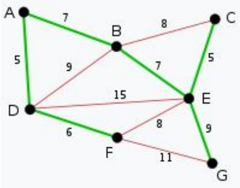
step 4



step 5



step 6



References for Kruskal's Algorithm

[1] Cormen, Thomas; Charles E Leiserson, Ronald L Rivest, Clifford Stein (2009). "Introduction To Algorithms" (Third ed.). MIT Press. p. 631. ISBN 978-0262258104.
[2] Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". Proceedings of the American Mathematical Society. 7 (1): 48–50.