Network Flows

UPCOPENCOURSEWARE number 34414

Topic 5:
Minimum Cost Spanning Tree

F.-Javier Heredia

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Estadística
i Investigació Operativa

5.- Minimum Cost Spanning Trees (Chap. 13 AMO)

- Definitions
- Applications
- Properties
- Algorithms:
  - Kruskal's algorithm.
  - Prim's algorithm.
  - Sollin's algorithm.
- Exercise.
- Source material:
  - R.K. Ahuja, Th.L. Magnanti, J. Orlin "Network Flows", chap. 13.
  - J. Orlin "Network Optimization" http://ocw.mit.edu/courses/sloan-school-of-management/15-082j-network-optimization-fall-2010/

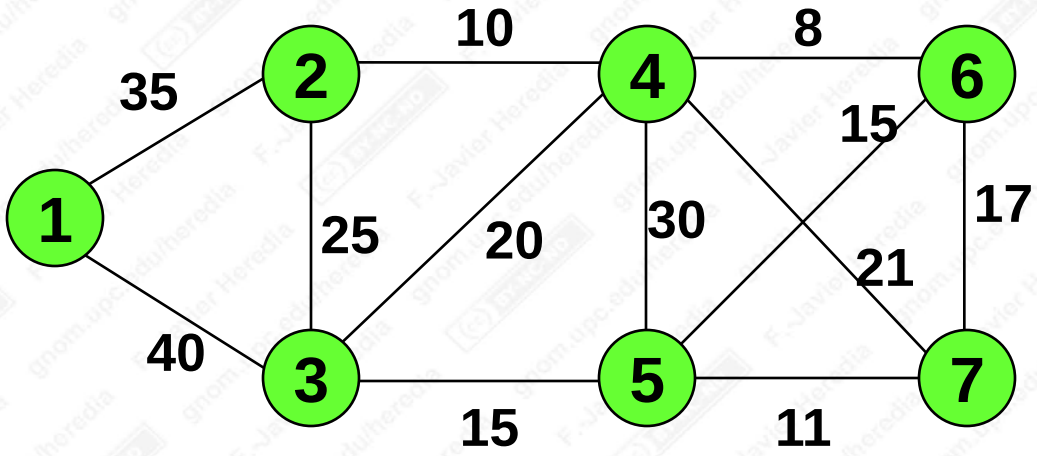
Minimum Cost Spanning Tree Problem

- Undirected network G = (N, A).
  - (i, j) is the same arc as (j, i).
  - We associate with each arc (i, j) $\in$ A a *cost* c_{ij}.
- A **spanning tree** T of G is a connected acyclic subgraph that spans all the nodes.
  A connected graph with n nodes and n – 1 arcs is a spanning tree.
- **The minimum cost spanning tree problem** is to find a spanning tree of minimum cost.

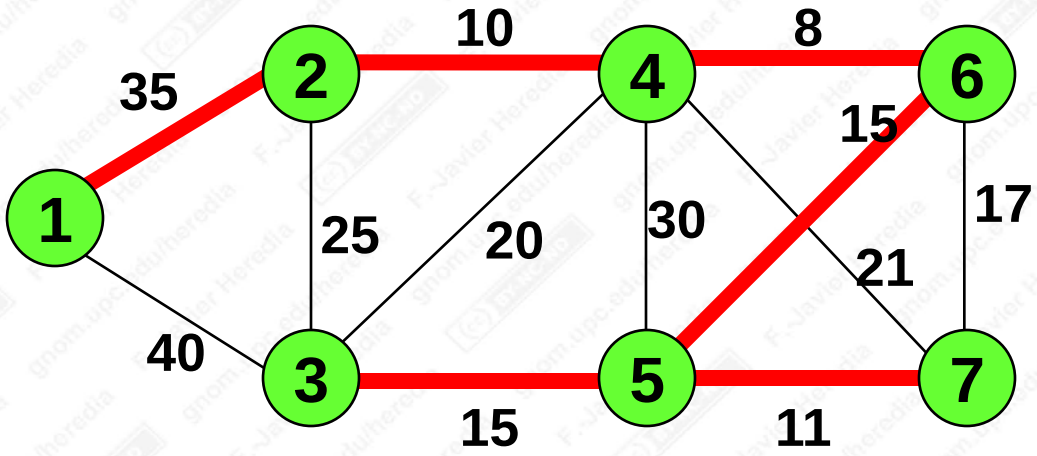
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Estadística
i Investigació Operativa

F.-Javier Heredia http://gnom.upc.edu/heredia

A Minimum Cost Spanning Tree Problem

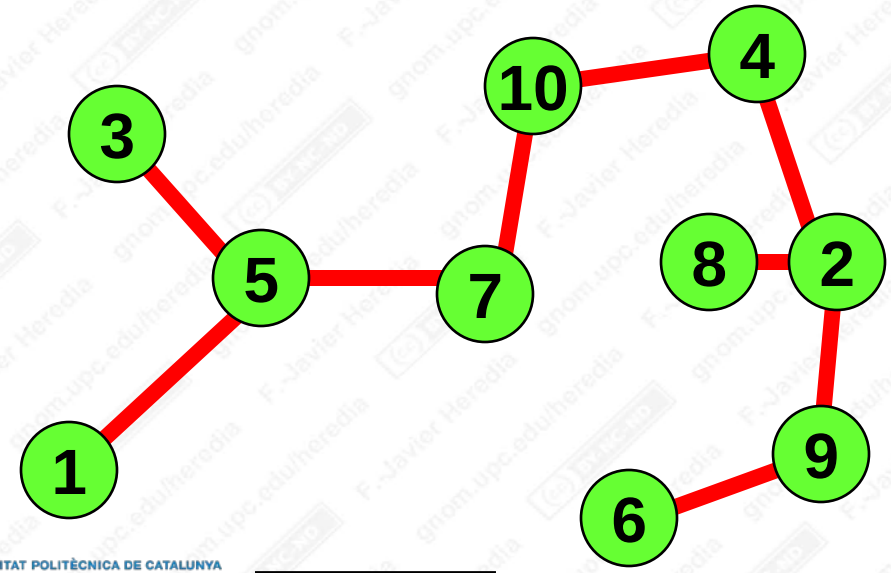


A Minimum Cost Spanning Tree



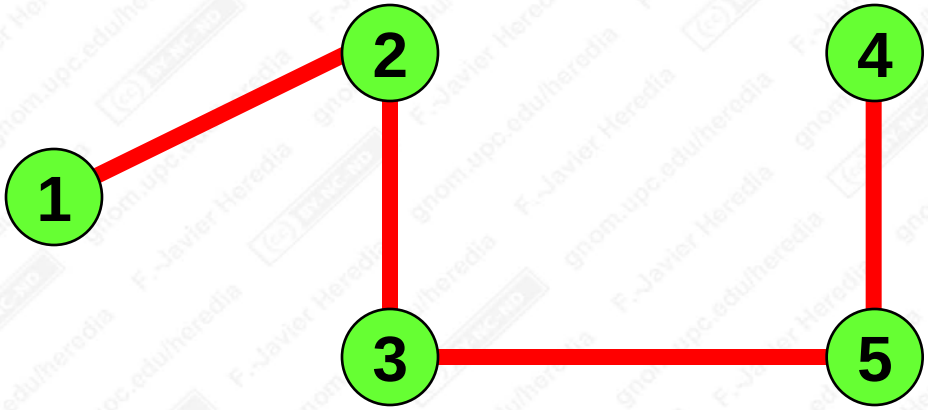
Communications Systems

- Consider a communications company, such as AT&T or GTE that needs to build a communication network that connects n different users. The cost of making a link joining i and j is c_{ij}. What is the minimum cost of connecting all of the users?



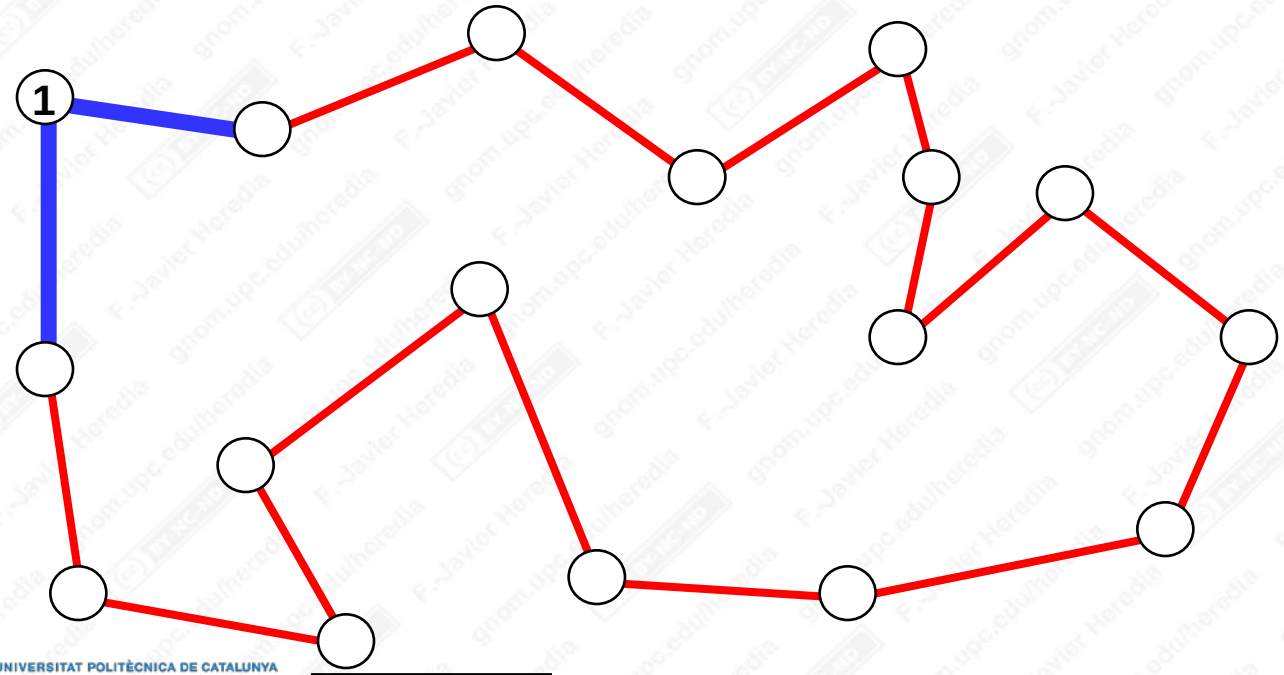
Electronic Circuitry

- Consider a system with a number of electronic components. In order to make two pins i and j of different components electrically equivalent, one can connect i and j by a wire. How can we connect n different pins in this way to make them electrically equivalent to each other so as to minimize the total wire length.



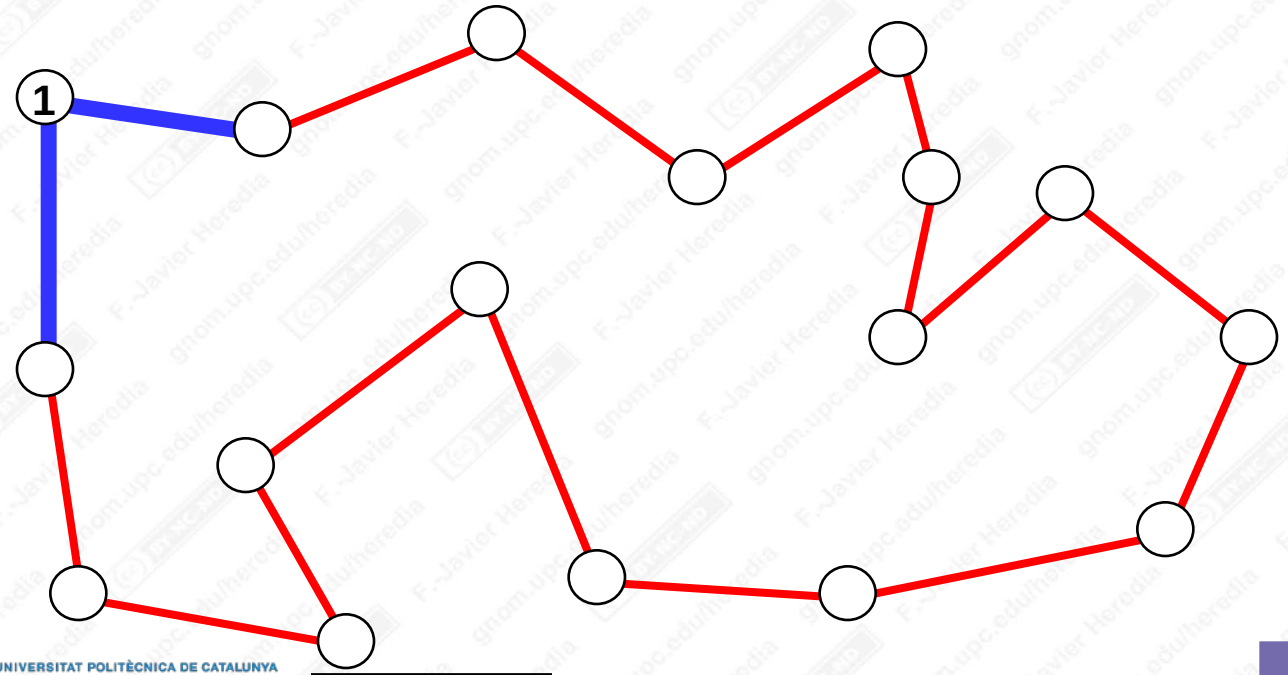
The Traveling Salesman Problem

- Consider the traveling salesman problem of finding a minimum cost tour linking n cities.
- One way of formulating this problem is using minimum spanning trees.



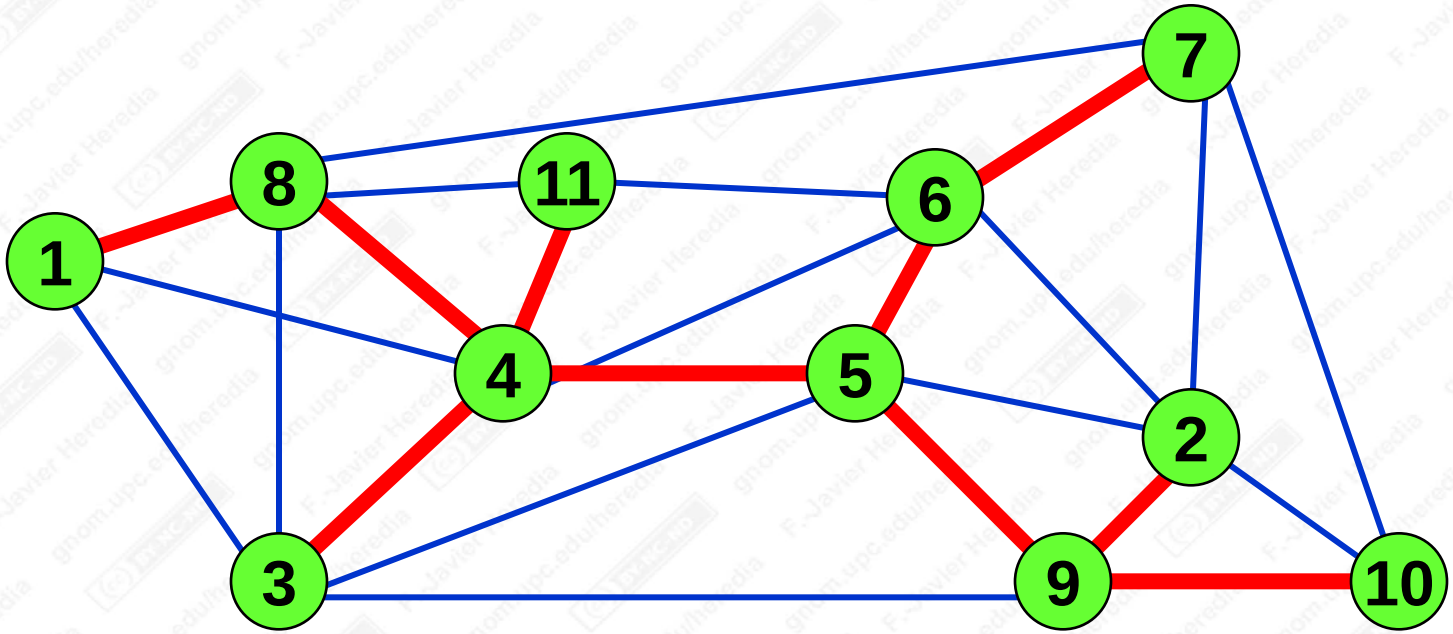
Representing the TSP problem

- A collection of arcs is a **tour** if
  1. There are two arcs incident to each node
  2. The red arcs (those not incident to node 1) form a spanning tree in G\1.
  3. The graph T shown is a **one-tree** (a tree with one cycle)
- **Tour ≡ one-tree + all nodes with degree 2**



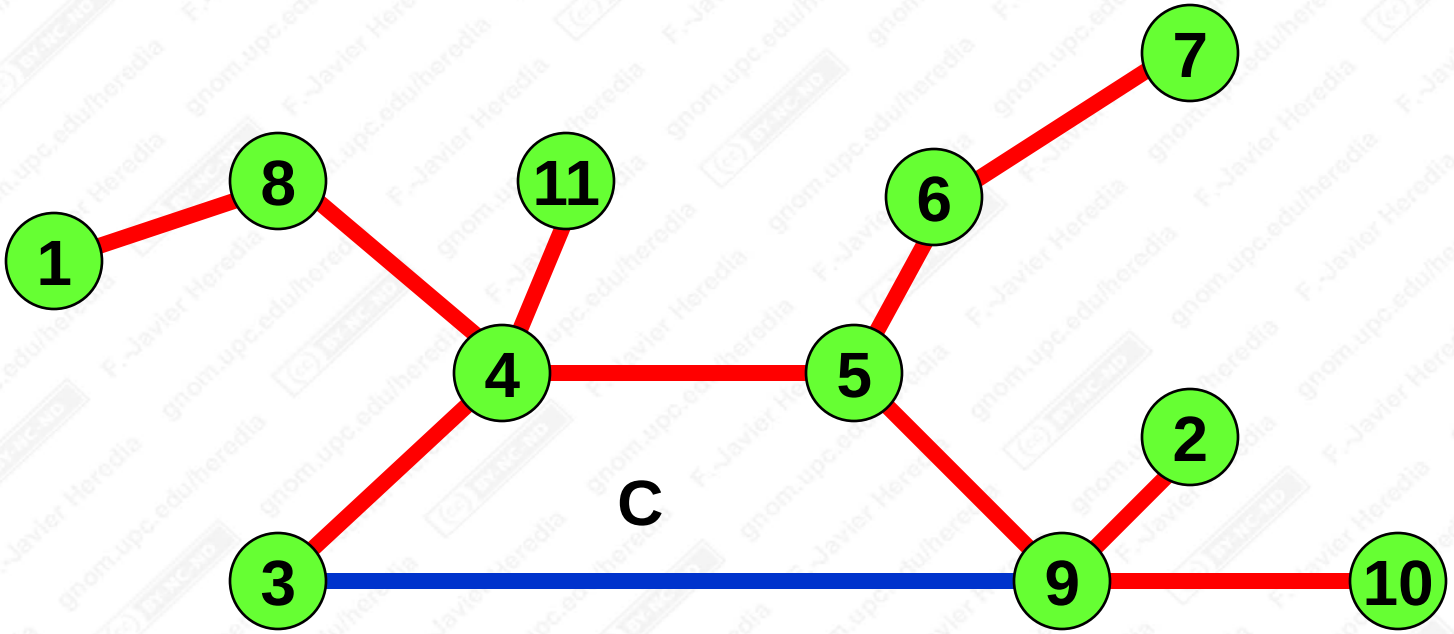
Definitions

- Let T* be a spanning tree. We refer to the arcs in T* as **tree arcs** and to those arcs not in T* as **nontree arcs**.



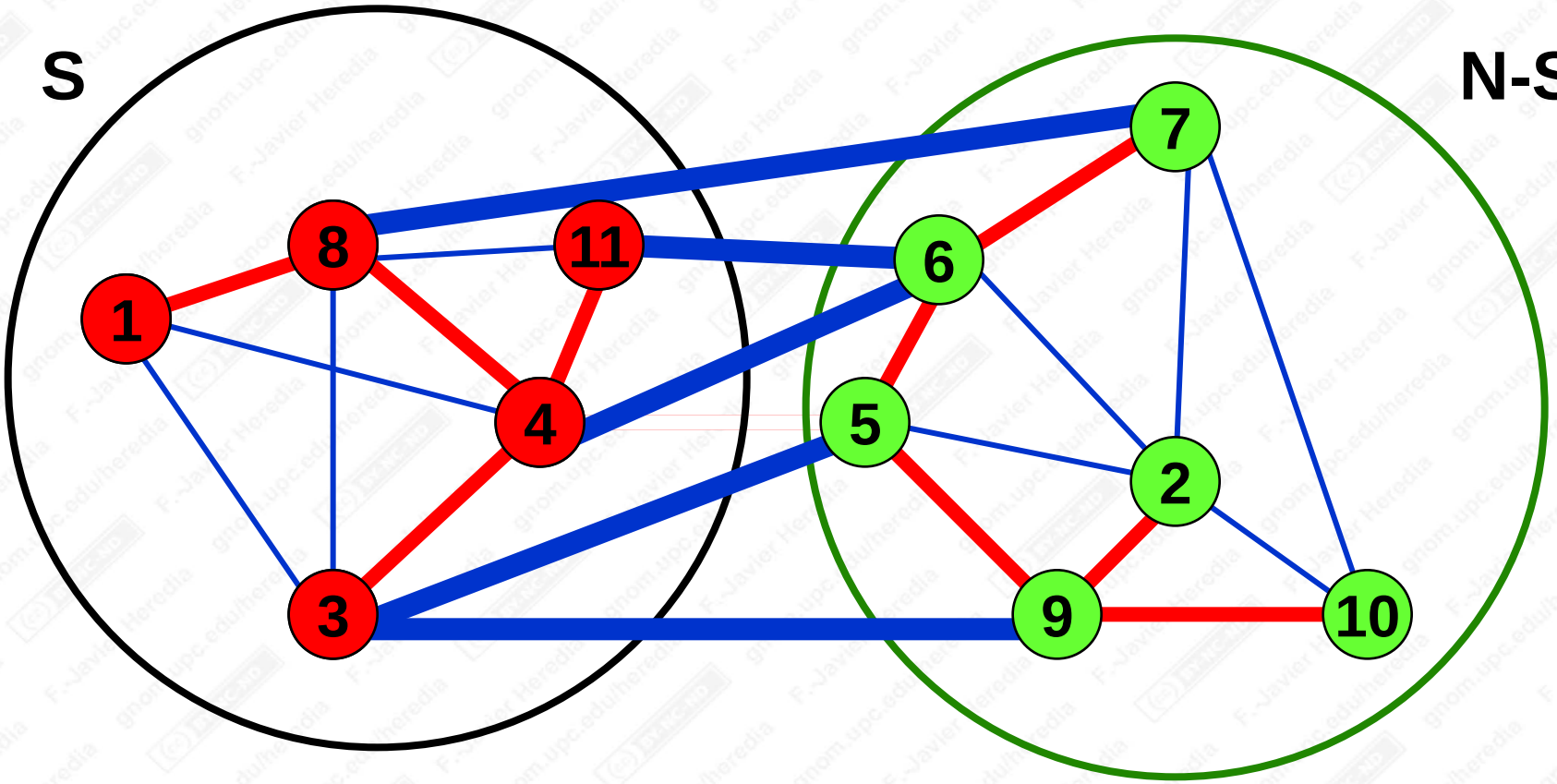
Lemma

Lemma: *Let T be any spanning tree. Let **a** be any nontree arc. Then adding **a** to T creates a unique cycle C. Deleting any arc of C creates a spanning tree T'. (left as exercise).*



Definitions

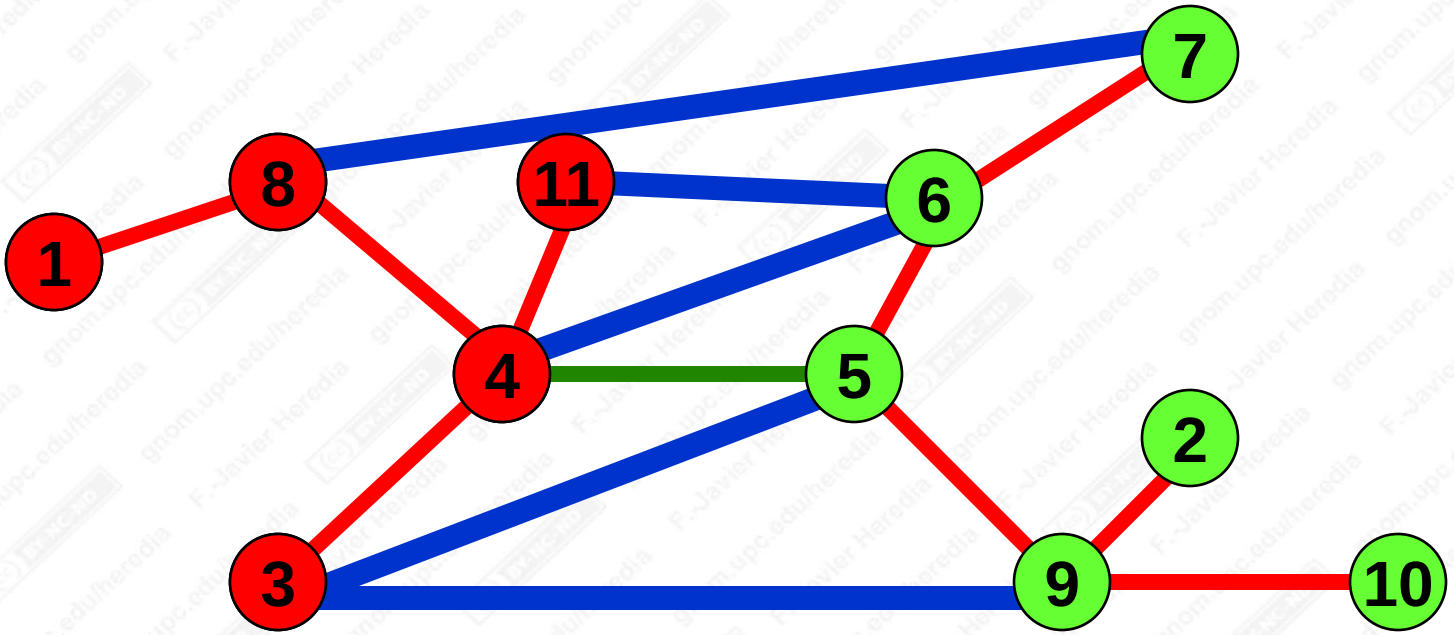
- **Def:** *If (i,j) is a tree arc, then T*-(i,j) has two components, say with node sets S and N-S. The arcs in (S,N-S) constitute a* ***cut****. For example, suppose that (i,j) = (4, 5).*



Cut Optimality Conditions

- **Def. (Cut optimality conditions):** *For every tree arc (i, j) $\in$ T* , $c_{ij} \leq c_{kl}$ for every arc (k, l) contained in the cut formed by deleting arc (i, j) from T*.*

Example: Consider the cutset induced by deleting (4,5). The cut optimality conditions are satisfied if every arc in the cut set has length at least that of (4,5).



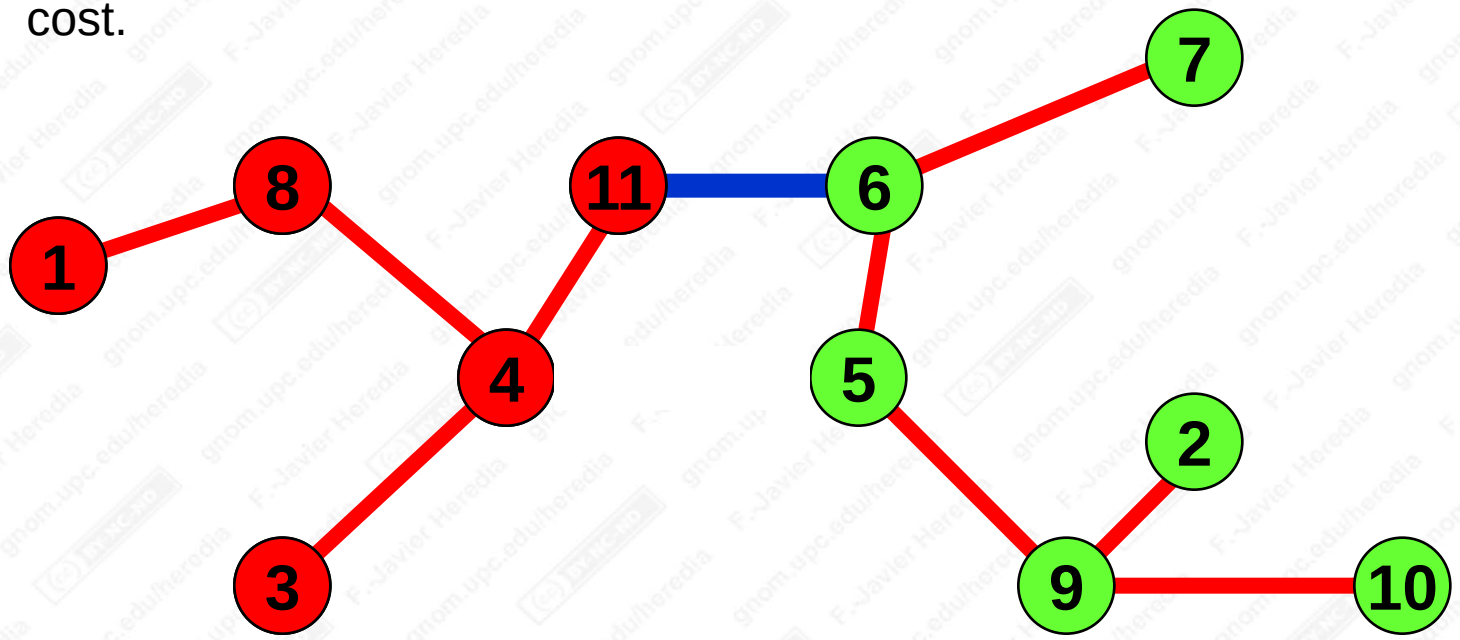
Cut Optimality Theorem

- **Theorem.** *A spanning tree T* is a minimum spanning tree if and only if it satisfies the* ***cut optimality conditions.***

Proof:

$\Rightarrow$

– Suppose first that $c_{ij} > c_{kl}$ for some arc (k,l) in the cutset obtained from deleting (i,j) from T*. Then T' = T*- (i,j) + (k,l) is a spanning tree with less cost.



Cut Optimality Theorem

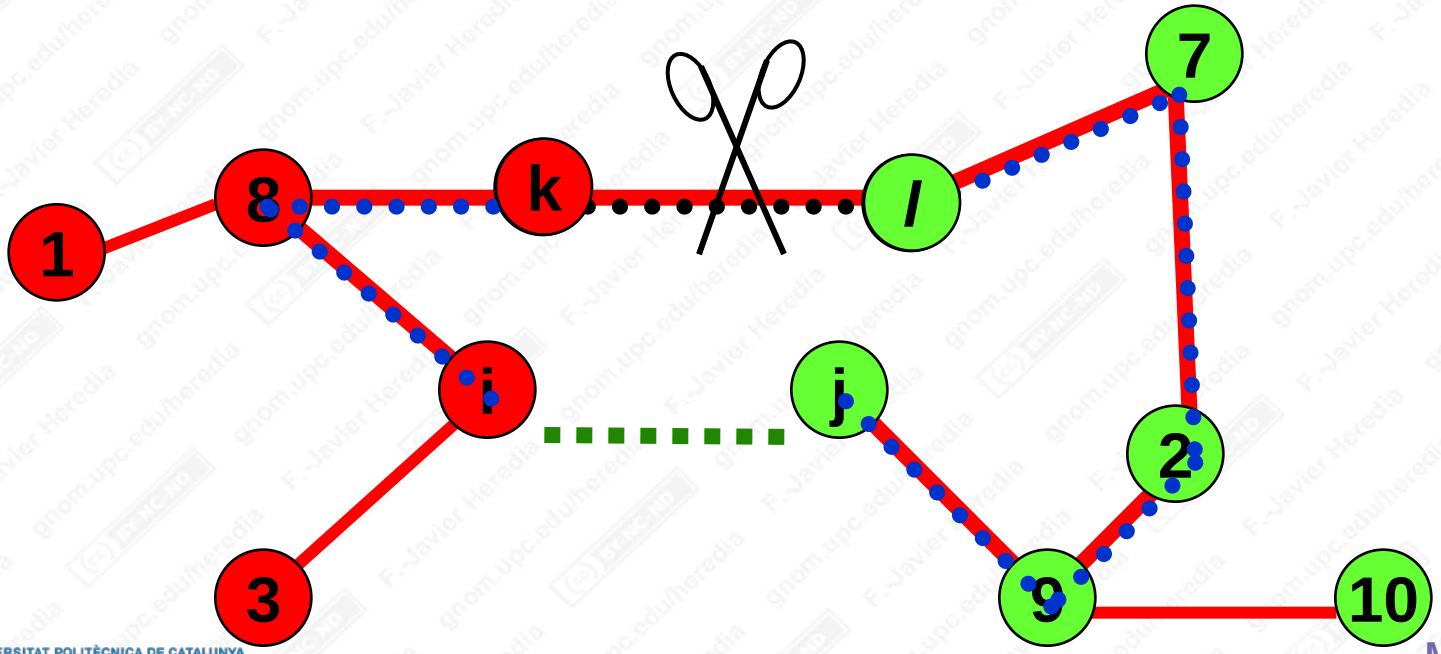
⇐

– Conversely, suppose that T* satisfies the cut optimality conditions (c.o.c.). Let T' be an optimal spanning tree, T'≠ T*. Let (i,j) ∈ T*, (i,j) ∉ T' . We claim that (i,j) ∈ T'. Suppose that this were not true.

– Let P be the path from i to j in T'. Then P has at least one arc (k,l) in the cutset induced by deleting (i,j) from T*.

– But then T' – (k,l) + (i,j) is also a min cost tree as $c_{ij} = c_{kl}$ (⇐ c.o.c.).

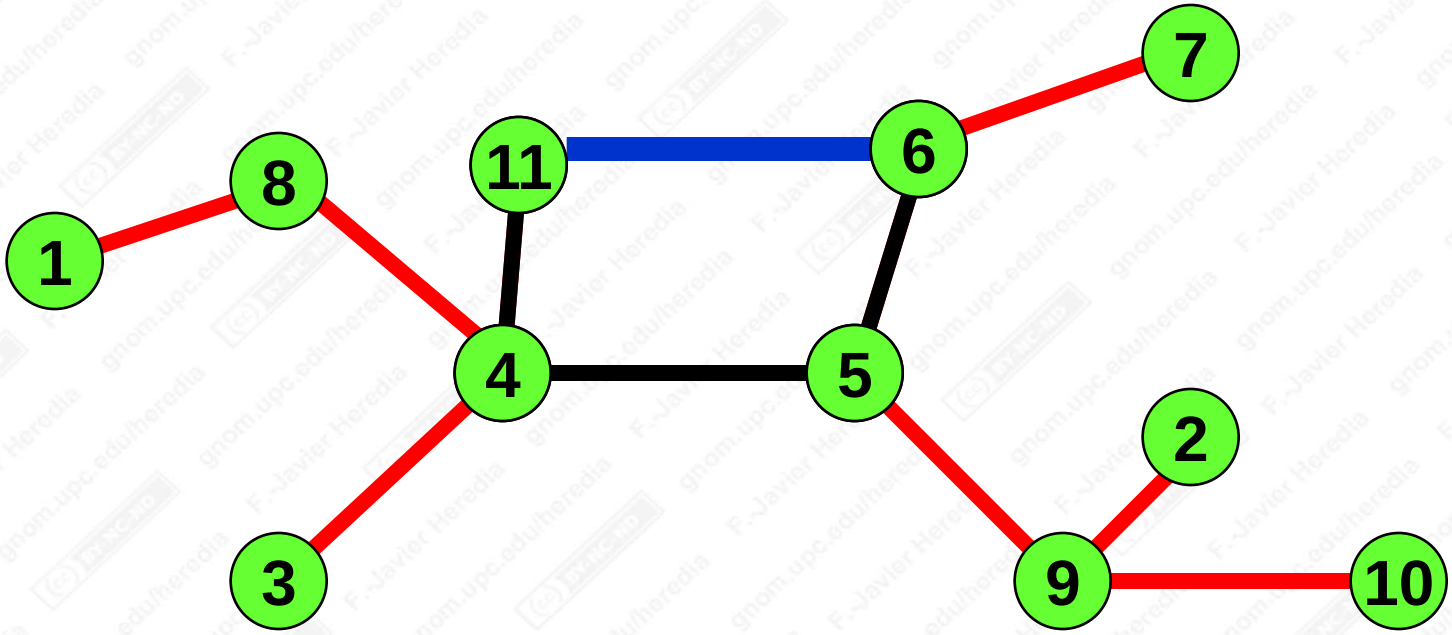
Repeating the process for all the arcs in T* we show that T* is also optimal ■.



Path Optimality Conditions

- **Def.(Path optimality conditions) :** *For every nontree arc* ***(k, l)*** *of G,* $c_{ij} \le c_{kl}$ *for every arc (i, j) contained in the path of* ***T**** *connecting nodes k and l.*

Example. Every arc in the path from 11 to 6 in ***T**** has length at most that of (11,6)

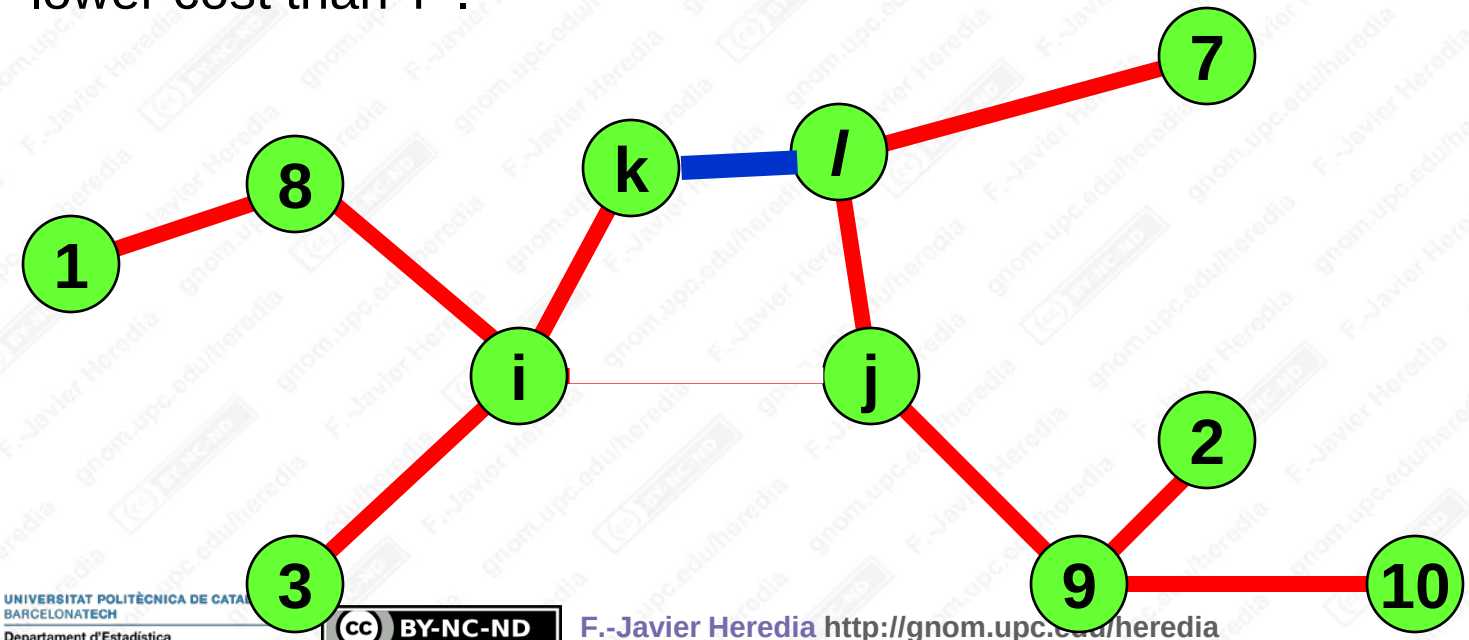


Path Optimality Theorem

- **Theorem 13.2. (Path optimality theorem)** : *A spanning tree **T*** is a minimum spanning tree if and only if it satisfies the path optimality conditions.*

Proof:

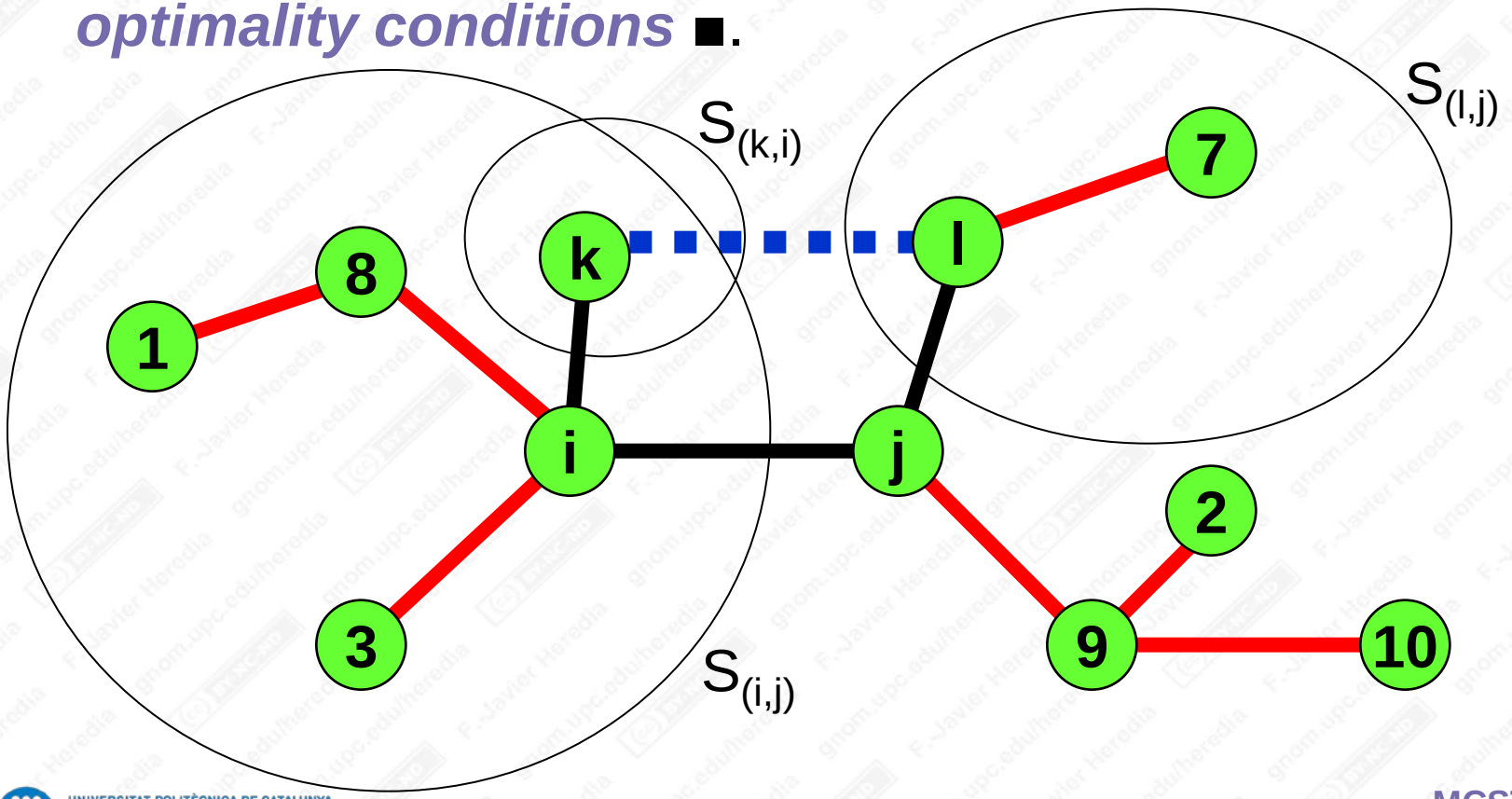
$\Rightarrow$: Suppose first that ***T**** is optimal and the path optimality conditions are not satisfied. Suppose $c_{ij} > c_{kl}$. Then T' = ***T**** - **(i,j)** + **(k,l)** has lower cost than T*.



Path Optimality Theorem

⇐ : **Observation**. Let (k,l) be an arc in A-***T****. Then (i,j) is on the cycle contained in ***T**** + (k,l) if and only if (k,l) is in the cutset induced by ***T****-(i,j).

Thus, ***path optimality conditions are equivalent to cut optimality conditions*** ■.



Kruskal's Greedy Algorithm

begin

order the arcs $a_1, \ldots, a_m$ in non-decreasing order of cost

$T = \emptyset$

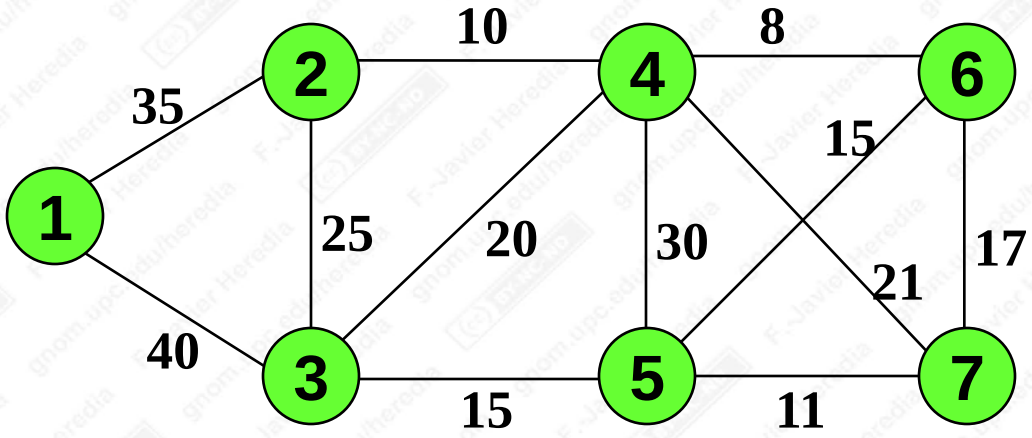
for i = 1 to m **do**

if $T + a_i$ does not have a cycle **then** $T := T + a_i$

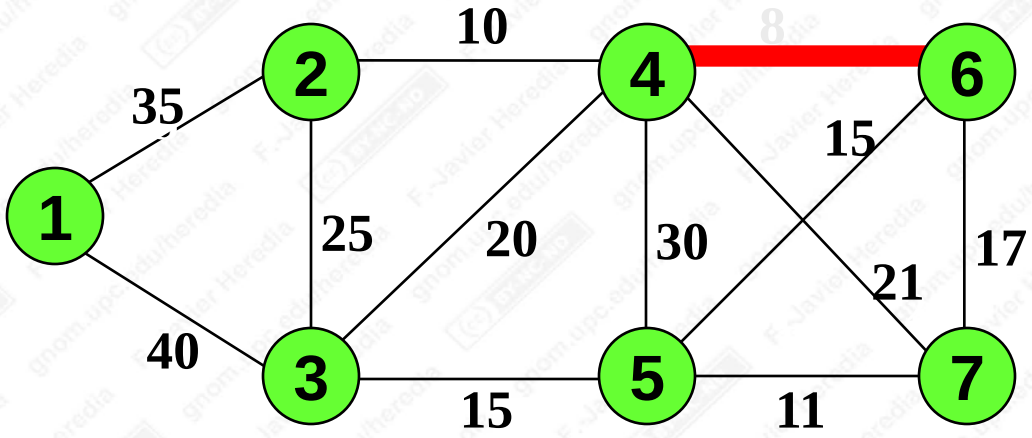
end

Correctness: at every stage k, the cost of the discarded arc (i,j) is ≥ than the costs of any arc in the path $P(i \rightarrow j)$ on $[T^*]^k \Rightarrow$ path optimality conditions.

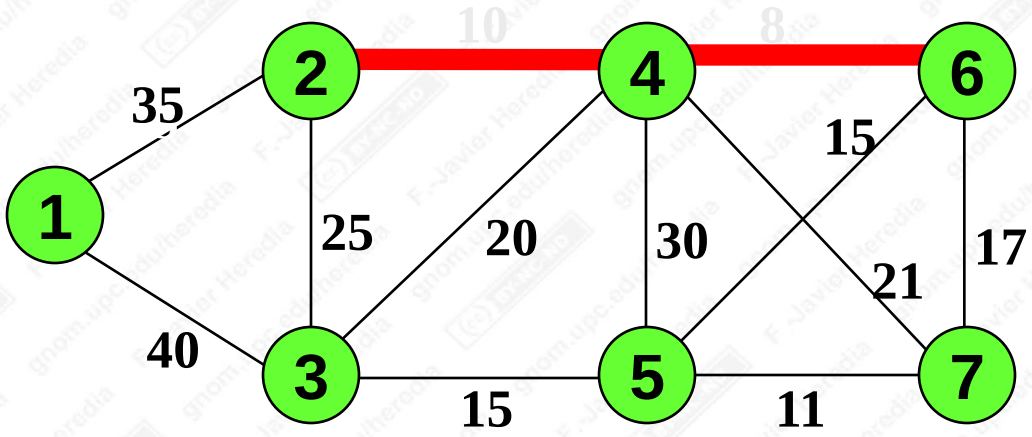
The Greedy Algorithm in Action



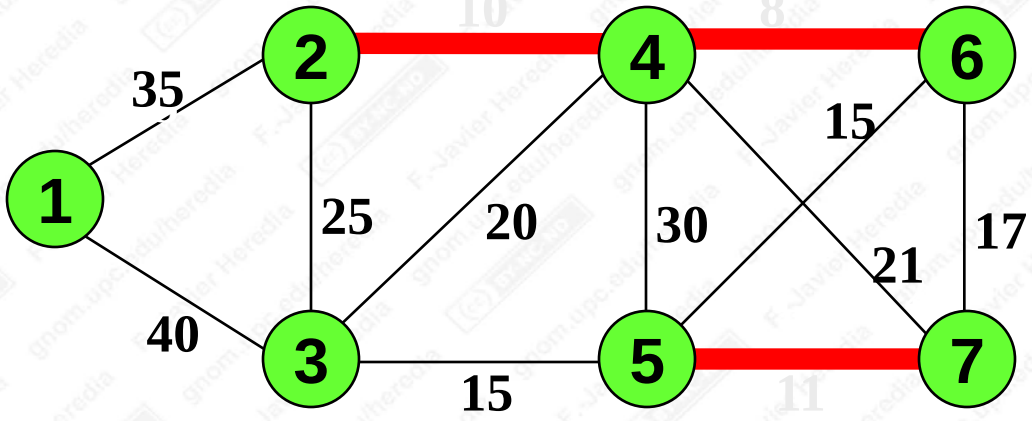
The Greedy Algorithm in Action



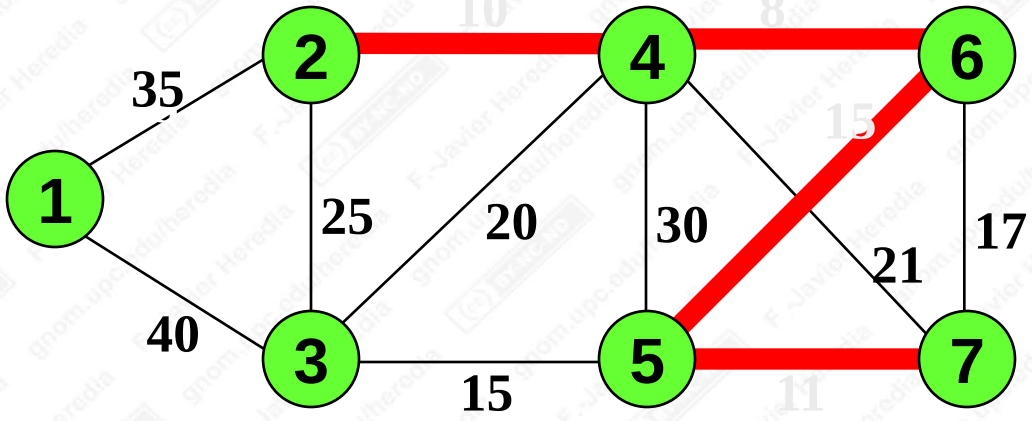
The Greedy Algorithm in Action



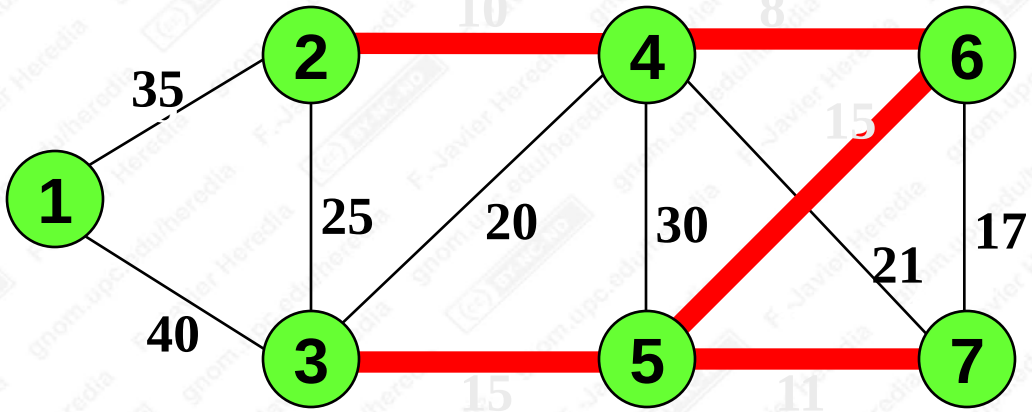
The Greedy Algorithm in Action



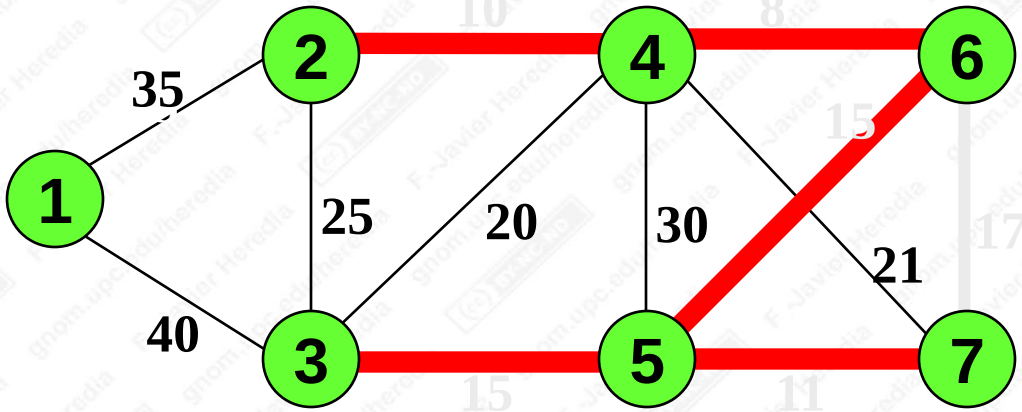
The Greedy Algorithm in Action



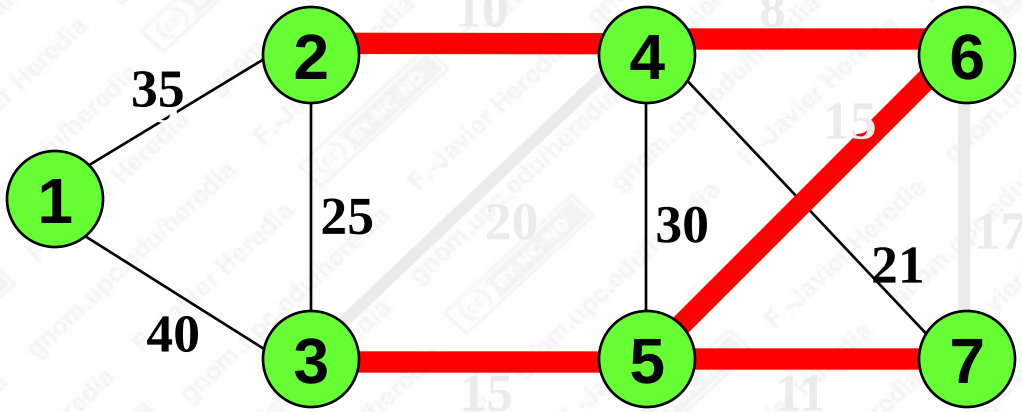
The Greedy Algorithm in Action



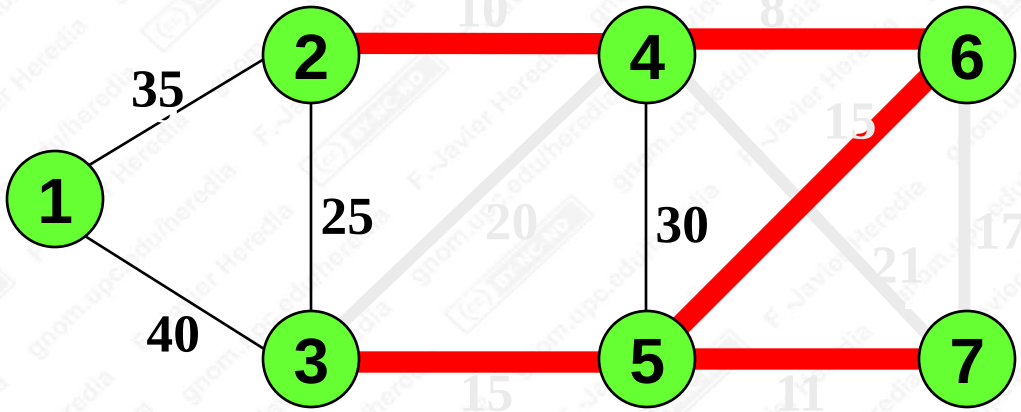
The Greedy Algorithm in Action



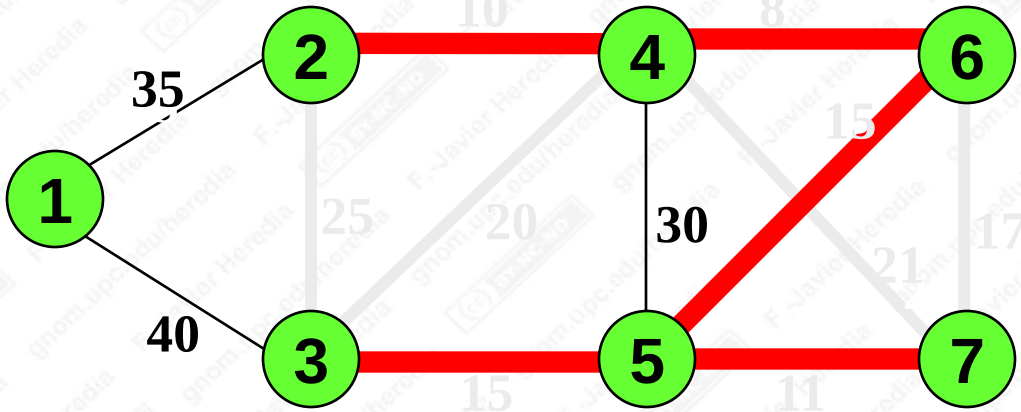
The Greedy Algorithm in Action



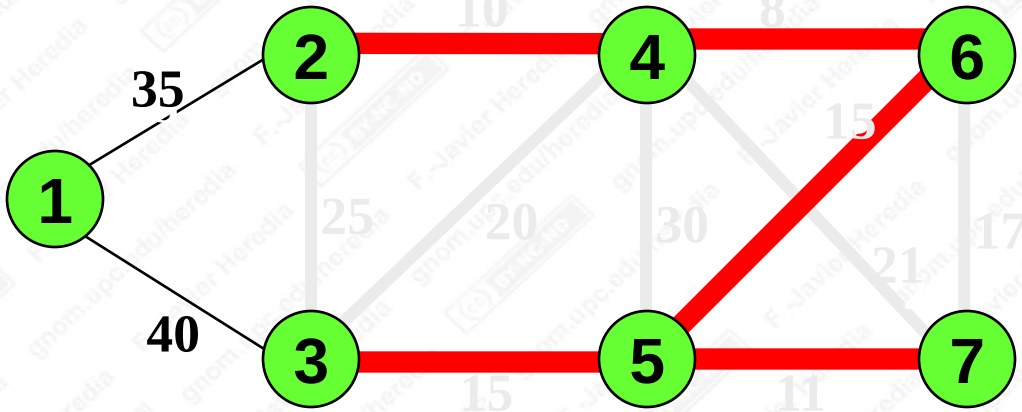
The Greedy Algorithm in Action



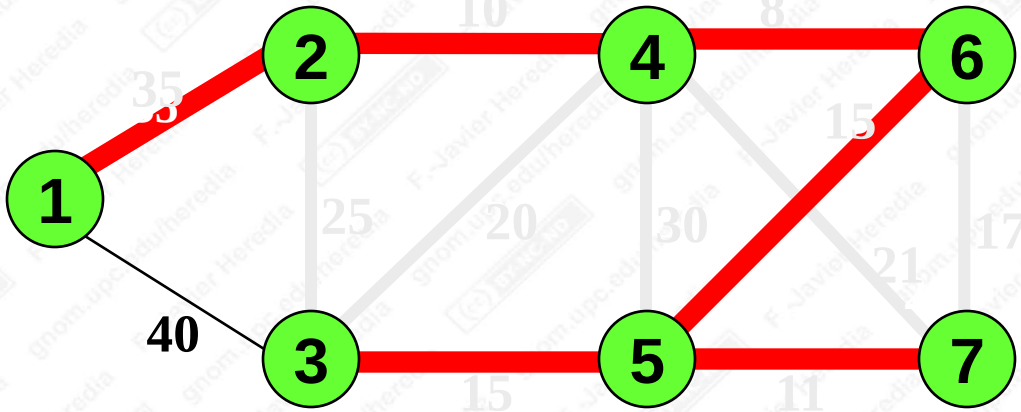
The Greedy Algorithm in Action



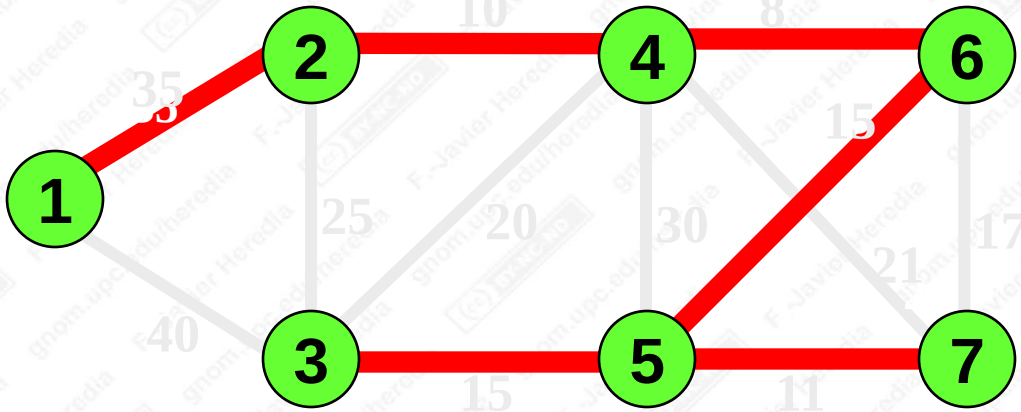
The Greedy Algorithm in Action



The Greedy Algorithm in Action



The Greedy Algorithm in Action



Running Time Analysis

- **Step 1**. Sort the arc costs.

  O(m log n)

- **Repeated Step**: determine if T+ a_i has a cycle?

  When arc (j,k) is considered, we want to know if j and k are in the same connected component C. If they are not, they result in two components, C and D, being merged.

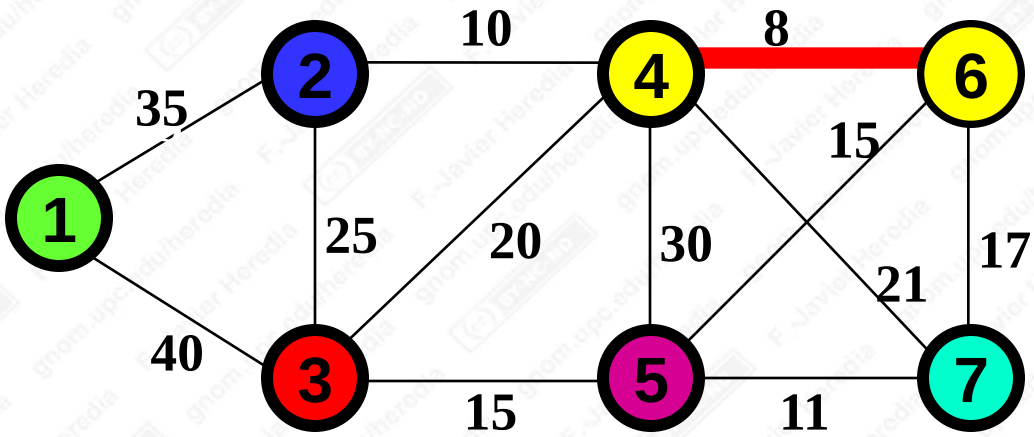
- Next: a very simple way of keeping track of components.

A simple, yet fairly efficient data structure

- Store each component C as set.
  - Let First(C) = first node in the component of C.
- For each element j in a component C, let
  - First(j) = First(C) = first node of C.
- Remark: adding (i,j) to a forest F creates a cycle if and only if (i,j) are in the same component; i.e.,
  - First(i) = First(j).
- When merging components C and D, put the larger component first. If |C| > |D|,
  - First(C∪D) := First(C).

The Greedy Algorithm with merging

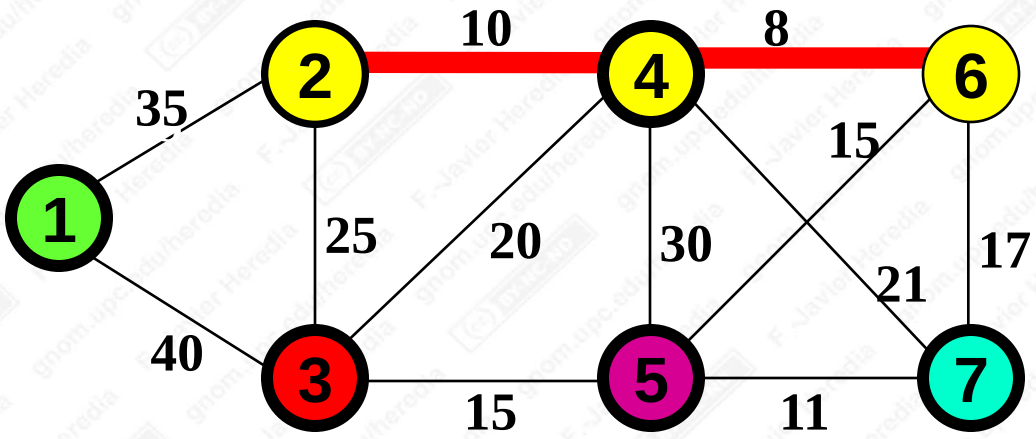
Node	1	2	3	4	5	6	7
First	1	2	3	4	5	4	7



root node

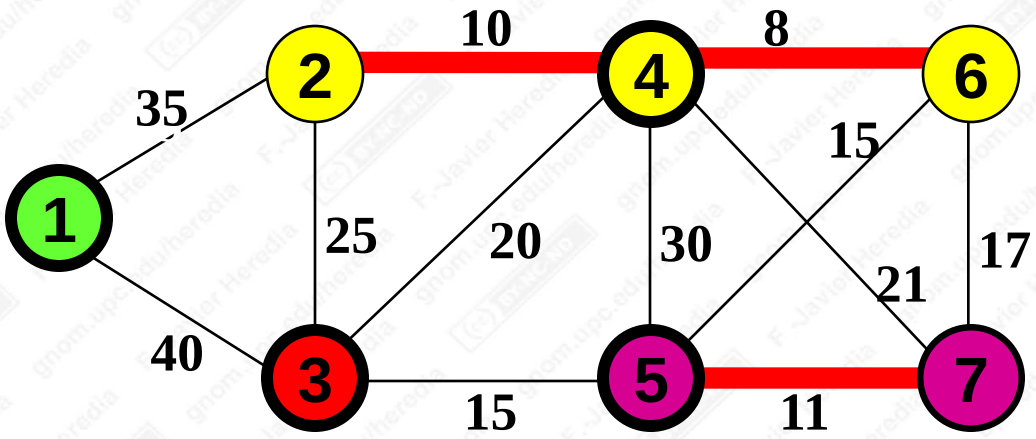
The Greedy Algorithm with merging

Node	1	2	3	4	5	6	7
First	1	4	3	4	5	4	7



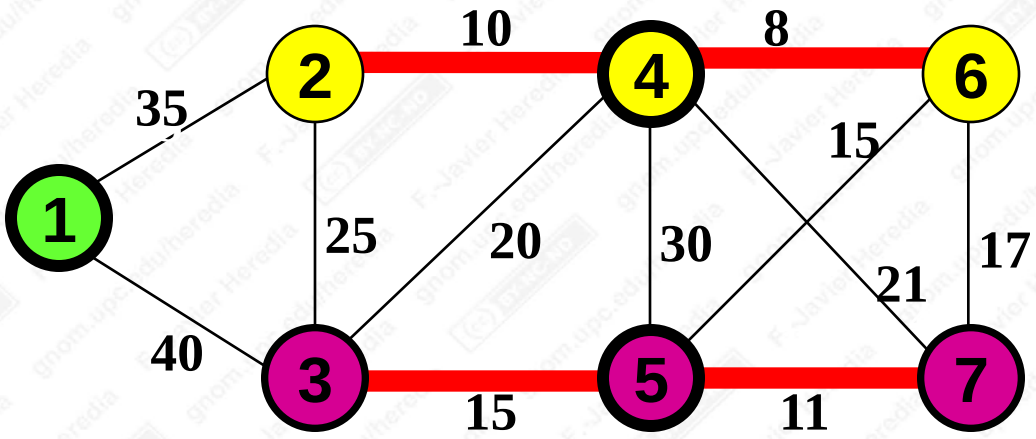
The Greedy Algorithm with merging

Node	1	2	3	4	5	6	7
First	1	4	3	4	5	4	5



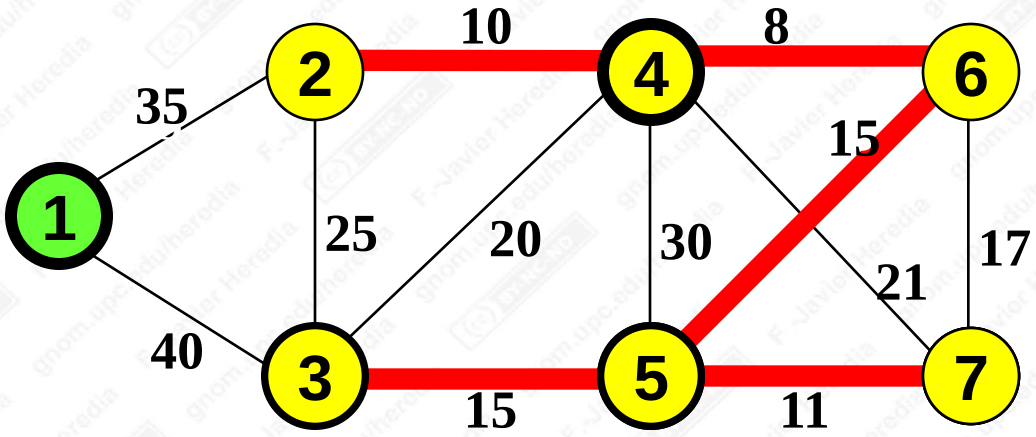
The Greedy Algorithm with merging

Node	1	2	3	4	5	6	7
First	1	4	5	4	5	4	5



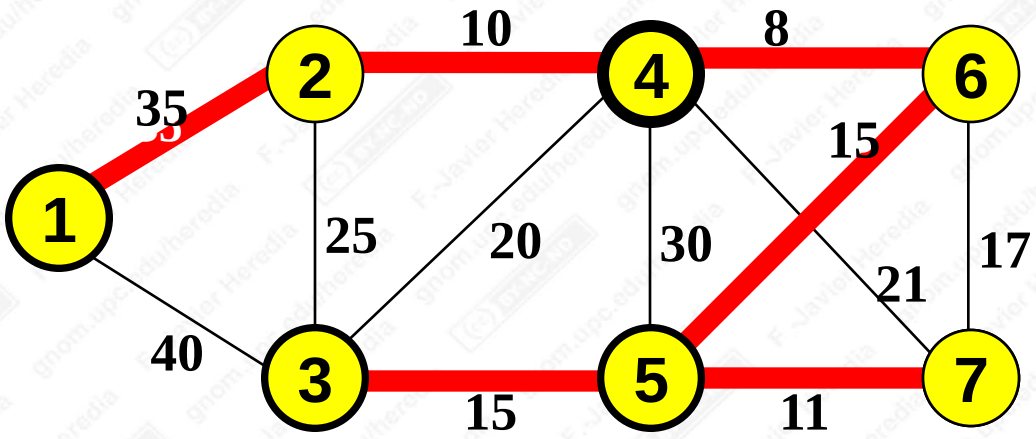
The Greedy Algorithm with merging

Node	1	2	3	4	5	6	7
First	1	4	4	4	4	4	4



The Greedy Algorithm with merging

Node	1	2	3	4	5	6	7
First	4	4	4	4	4	4	4



Analysis of the Running Time

- Time to determine if First(i) = First(j) for i,j:
  - O(1) per arc. **O(m)** in total.
- Time to merge components C and D:
  - Assume $|C| \geq |D|$.
  - O(1) per node of D (the smaller side)
  - Each node *i* is on the smaller side of a merge at most log n times. (Because the number of nodes in the component containing *i* at least doubles when it is merged.)
  - Total time spent merging: **O(n log n).**
- Total running time.

$$\mathbf{O(m + n \log n + m \log n)}$$

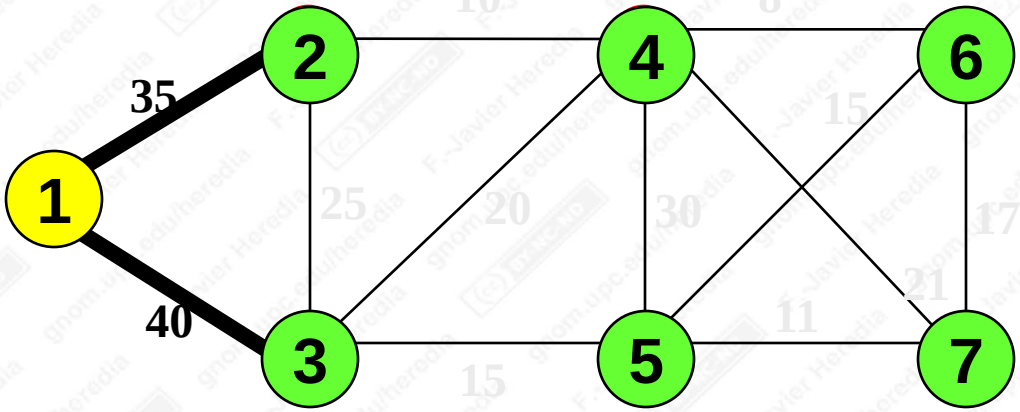
Prim's Algorithm, a Dijkstra-like Algorithm

Exemple

```
begin
    S = {1};   T = ∅;
    while S ≠ N do
    begin
        find the minimum cost arc (i,j)  from S to N-S;
        add j to S; add (i,j) to T;
    end
end
```

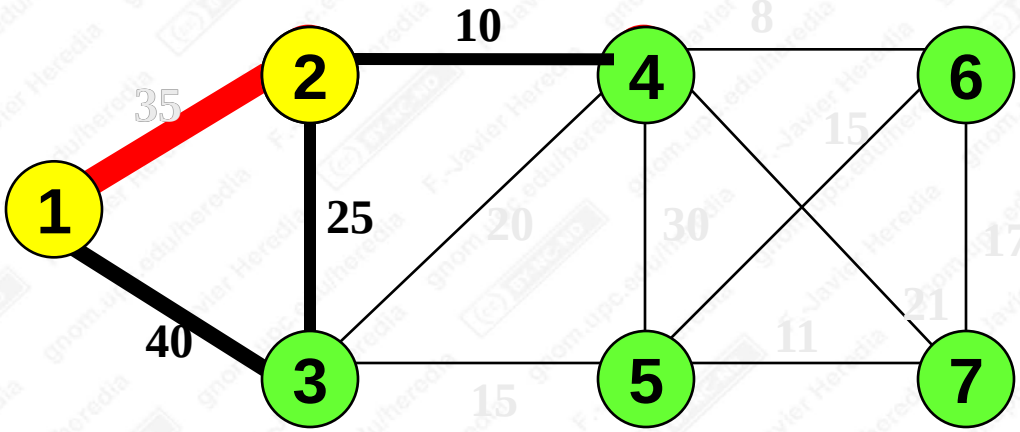
Correctness: This algorithm builds a minimum cost spanning tree from node 1. The algorithm guarantees that, by construction, **each arc added to T satisfy the cut optimality conditions** which are satisfied at the end.

Prim's Algorithm in Action

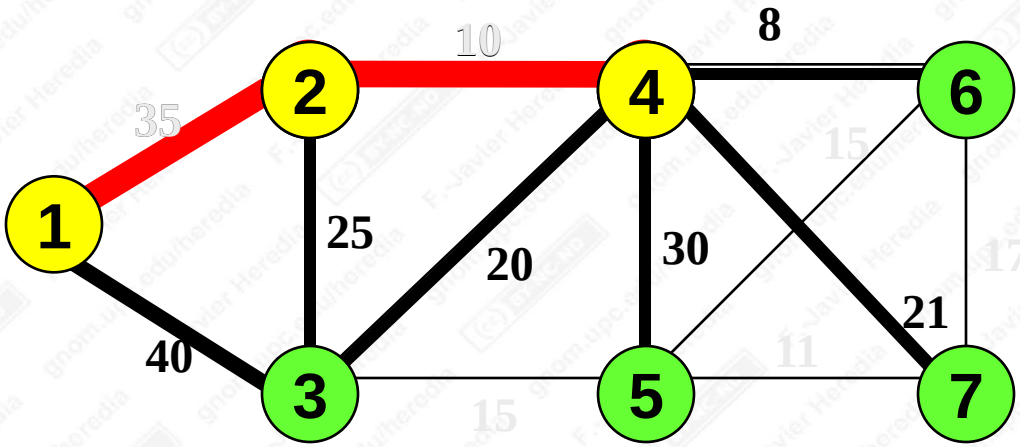


The minimum cost arc from yellow nodes to green nodes can be found by placing arc values in a priority queue.

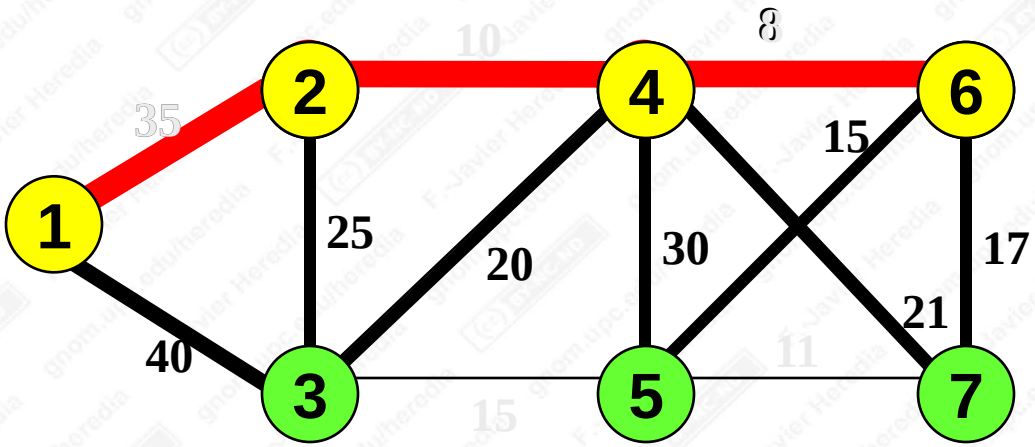
Prim's Algorithm in Action



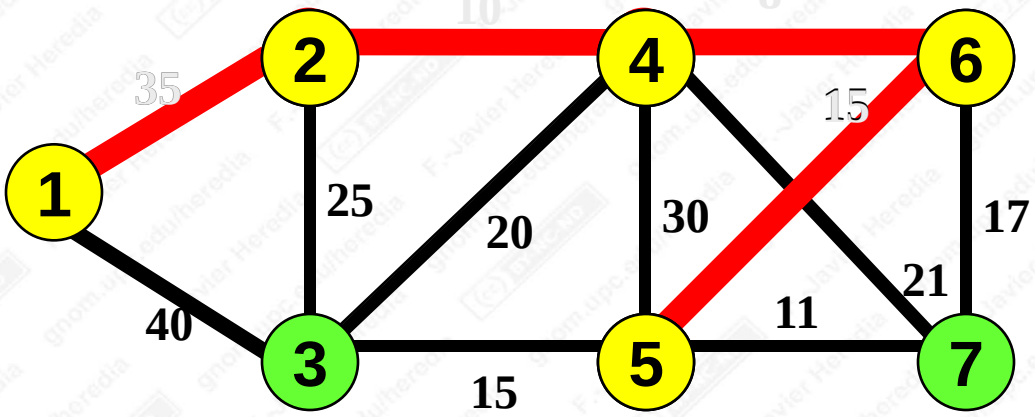
Prim's Algorithm in Action



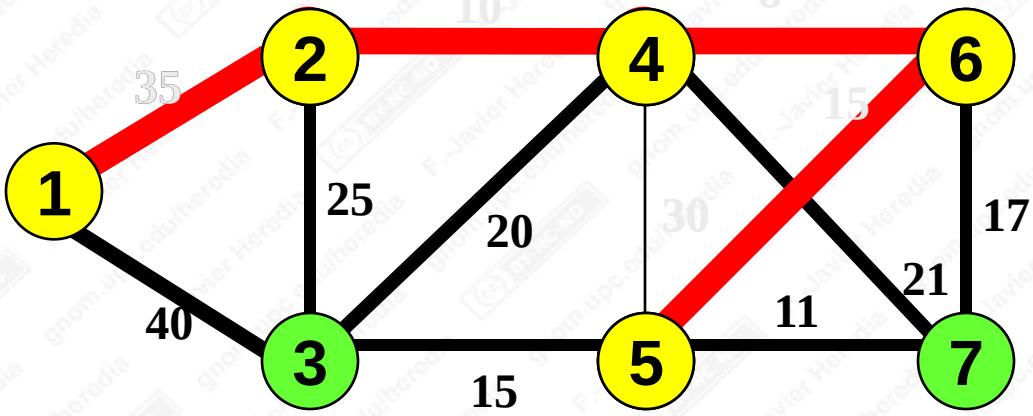
Prim's Algorithm in Action



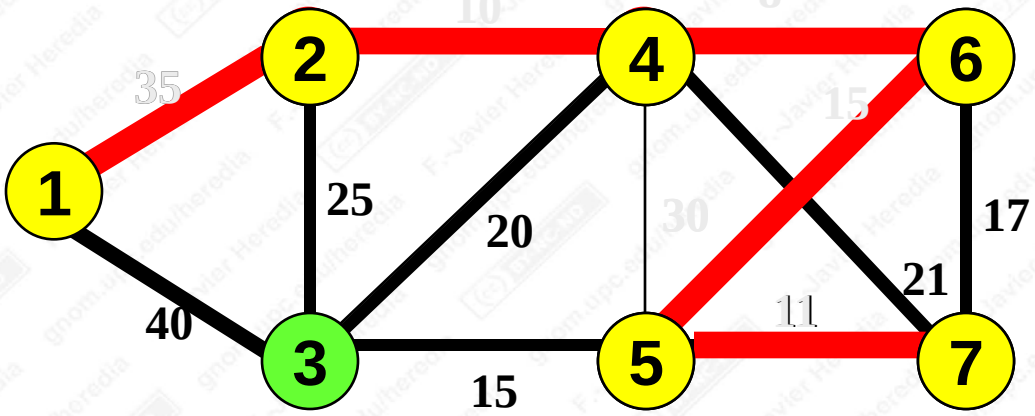
Prim's Algorithm in Action



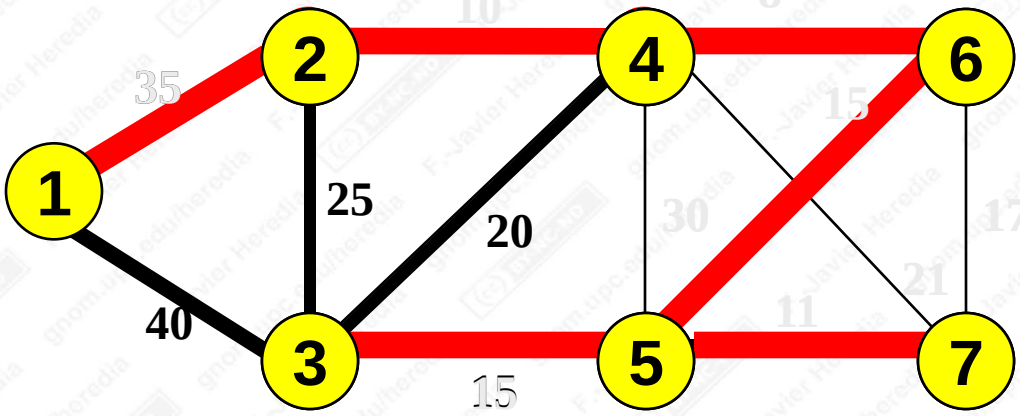
Prim's Algorithm in Action



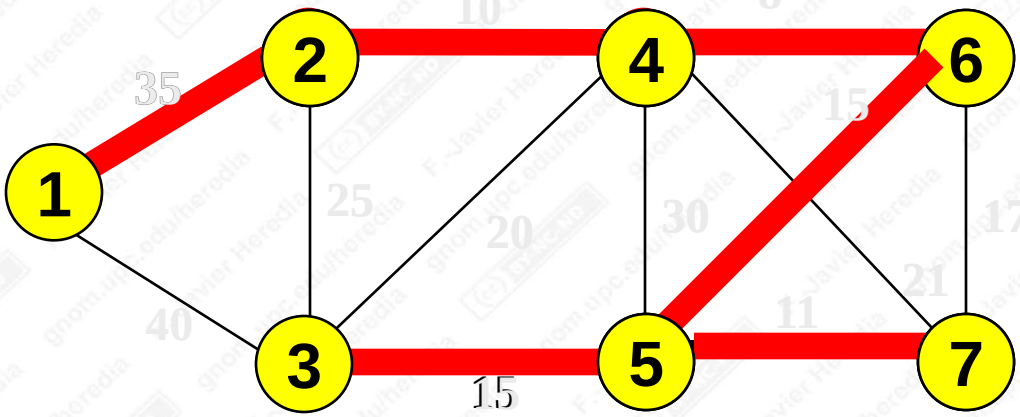
Prim's Algorithm in Action



Prim's Algorithm in Action



Prim's Algorithm in Action



Running time of Prim's Algorithm

- n-1 iterations.
- At each iteration: select the minimum cost arc in [S,N-S]$\rightarrow$ at most m operations:
- Total running time of Prim's algorithm: **O(nm).**
  - Improved with heap implementations: O(m log n)
  - Kruskal O(m + n log n + m log n) > O(m logn)

Sollin's Algorithm

begin

  for each $i \in N$ **do** $N_i := \{i\}$;

  $T^* := \varnothing$

  while $|T^*| < n - 1$ **do**

  begin

    for each tree N_k of the forest **do**

      find the least cost arc (i_k, j_k) from N_k to $N \backslash N_k$;
      in case of ties, use a consistent tie breaking rule

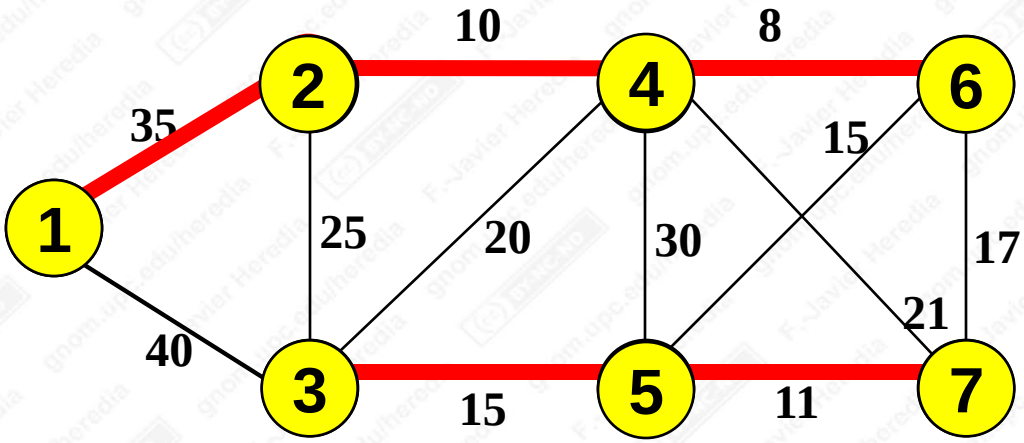
    let $T^* = T^* \cup \{(i_1, j_1), (i_2, j_2), \ldots, \}$

  end

end

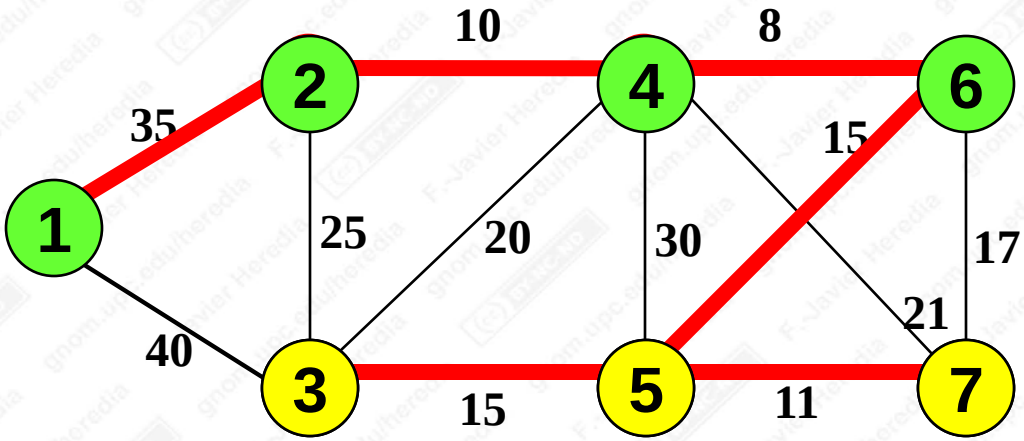
- **Correctness:** relies on the cut optimality conditions.
- **Running time:** at each iteration, the size of the smallest component at least doubles, and so there are O(log n) iterations. A straightforward implementation runs in O(m log n) time (AMO, Theorem 13.7) (same of Prim's alg.).

Sollin's Algorithm in Action



Treat all nodes as singleton components, and then select the min cost arc leaving the component.

Sollin's Algorithm in Action



Find the min cost edge out of each component

Exercises

- Consider the minimmum cost spanning tree problem over the two graphs on figure 13.15, page 537 of AMO:
  1. Select one of the graph and solve the MCSTP with the Kruskal's, Prim's and Sollin's algorithms.
  2. Check, for some arcs of T*, the satisfaction of the Cut Optimality Conditions.
  3. Check, for some arcs of A-T* the satisfaction of the Path Optimality Conditions.
  4. Verify your solution with the help of Giden.
- You must upload to the intranet a document (.doc or .pdf) with the response to the above questions and the file .gdn.